Java SMS SDK 2.6.0 Manual

Abstract

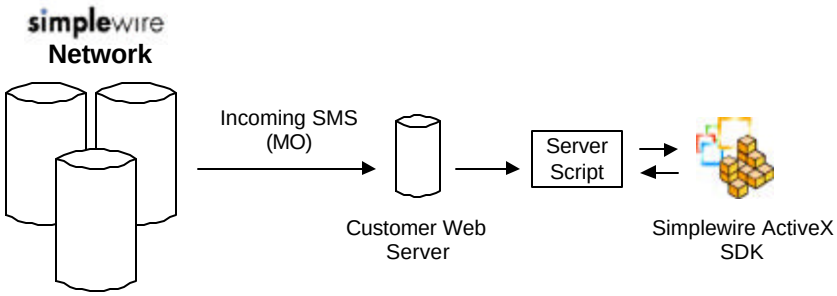
This document describes the methods and properties of the Simplewire™ Java SMS Software Development Kit.

Table of Contents

Introduction

The Java SMS SDK provides easy, high-level control of the Simplewire wireless messaging platform. The Java SMS SDK was designed be to be as developer-friendly as possible by hiding the intricacies of the XML format required to communicate with the Simplewire WMP (Wireless Message Protocol) Server(s). The Java SMS SDK makes it possible to send a wireless message with as little as two lines of code.

2-Way



With the release of Version 2.6, the Java SMS SDK can now support 2-Way messaging via the Simplewire Network. It is now possible to receive a wireless message with as little as four lines of code.

Getting Started

Getting started with the Simplewire Java SMS Software Development Kit only takes a few steps.

Simplewire Developer Program

If you have not already done so, please join the free Simplewire Developer Program. The program offers you the opportunity to create and quality-assure your wireless messaging application before commercially releasing it. The program provides wireless development tools and resources to help you get started at no cost.

Furthermore, you will gain access to an online account, from which you may manage all of your resources, as well as the Simplewire Virtual Mobile Phone, to which you may send free test messages online with your SMS Software Development Kit. To join the program, or for more information, visit http://devprogram.simplewire.com/ .

The Simplewire Developer Program now offers 2-Way Virtual Numbers. To receive your own virtual 2-Way you must register with the Simplewire Developer Program and then contact your local Simplewire Sales Representative. Your representative can provision a second Virtual Number with 2-Way connectivity.

30-Day Trial Messaging Account

In order to send live messages to actual mobile devices from your Developer Account, you must purchase Demo Messages, in the form of credits, on the Simplewire Wireless Messaging Network. Please note that Demo Messages will include the "Simplewire Evaluation" tag at the beginning of each message. All remaining Demo Message Credits will be automatically transferred to your commercial account when you Go Live! In order to purchase Demo Message Credits, login to Your Account at http://www.simplewire.com/account/, and click the "Demo It" button.

For 2-Way, demo credits are not available because you will need your own mobile number hosted on Simplewire's Network. Simplewire supports 2-Way numbers for many different countries and area codes. Please contact Simplewire for more information.

Commercial Messaging Account

In order to eliminate the "Simplewire Evaluation" tag and gain access to the full spectrum of features, you must upgrade to a commercial account and Go Live! You may do this by purchasing

a commercial license for the SMS Software Development Kit or an SMS Starter Package in the Simplewire Store located at http://www.simplewire.com/store/platform/.

Sales

You can contact our sales department at any time, by sending us a request from the web at http://www.simplewire.com/contact/sales/.

Installation

Java Compatability Note

This version was compiled with the JDK version 1.4.0. The swsms.jar file should run with any JDK 1.2 compliant runtime.

Using Packages

Note that the Java SMS SDK is 'packaged' and therefore must be installed properly according to CLASSPATH package rules in order to function properly. If you are not familiar with using Java packages (i.e. 'com.simplewire.sms' package), then please refer to your Java API documentation and/or tutorials to learn more.

Installing the JAR File

This installation assumes you have installed the JDK (Java Developer Kit), or an equivalent product.

Distribution files are packaged in ZIP format (.zip) or TAR.GZ format. Use your favorite ZIP tool to unpack the zip files. If you don't have a ZIP extractor, a Windows specific version can be downloaded at http://www.winzip.com.

You have two choices on how to install the swsms-X.X.X.jar file. You can either add the .jar file to your JDK or add it manually to your CLASSPATH variable.

Java 2 (aka JDK 1.2.x, 1.3.x, 1.4.x)

Copy swsms-X.X.X.jar into your $JavaLib/ext directory.

For example: for the JRE the default directory would be
(on Win32) "C:\Program_Files\JavaSoft\JRE\1.3.1\lib\ext".
(on Win32) **Note:** With JDK1.4.0 the JRE folder changed to "C:\Program_Files\Java ".
(on RH Linux 7.3) /usr/java/jdk1.3.1_03/jre/lib/ext

Note: By default, when using the JDK on Windows 2000 the JRE directory is used when running Java programs. If you put the JAR file into C:\j2sdk1.4.1\jre\lib\ext you will get java.lang.NoClassDefFoundError for the Java SMS SDK classes when running Java programs. This is because the JRE classpath by default points to the "Program Files\java\j2re1.4.1" directory ("Program Files\javasoft..." on JDK 1.3.x). If your going to use the Java

compiler (javac) put a copy of the JAR into C:\j2sdk1.4.1\jre\lib\ext this is the one the compiler will use.

Note: You will find that using CLASSPATH with JDK 1.3 and later is somewhat smarter than with previous JDKs. The classpath override appends the built-in CLASSPATH. For example when you use java classpath c:\jdev\jsp, that is the same as java classpath c:\jdk1.2\jre\lib\rt.jar;c:\jdev\jsp with JDK 1.2.

Note: If your using Visual Cafe 4, this folder would be something like C:\VisualCafe4\Java2\lib\ext. If you installed JDK 1.2.2 on Linux in /usr, the directory would be /usr/jdk1.2.2/jre/lib/ext. For Tru64 Unix this would typically be /usr/opt/java122/jre/lib/ext.

Example Code

Example code utilizing the Java SMS SDK is provided for Java applications with the install package. In order to view the example code, you will need to navigate to the directory where you installed the Java SMS SDK. Inside that directory, there will be an “examples” directory with all the example code listed above.

If you require further assistance, please visit Simplewire Code Central at http://www.simplewire.com/developers/code/.

Support

Please submit any problems, bug reports, incompatibilities, requests for change, or other comments to Simplewire Support. All bug reports should be accompanied by one or more concrete examples that will help us reproduce the problem. Include all relevant information that you think will help us recreate the particular environment in which the bug was observed. Remember, if we cannot reproduce the problem, we cannot fix it!

Architecture

There are three classes in the Simplewire Java SMS Software Development Kit: SMS, SMSCarrier, and SMSCarrierList.

2-Way

To utilize the SDK's 2-Way capabilities, you will make use of the XMLParseEx() method. This method should be passed in an XML string that is posted to a URL on a server you are hosting. Once you pass the incoming XML string into the SDK, it is parsed and the appropriate properties such as MsgCallback, MsgPin, and MsgText are populated.

Please see the XMLParseEx() method for more information.

SMS Class

Syntax

```
SMS sms = new SMS();
```

Remarks

The SMS object is used for setting all the properties needed to send a wireless message.

SMSCarrier Class

Syntax

```
Iterator i = sms.getCarrierListIterator();
SMSCarrier carrier = (SMSCarrier) i.next();
```

Remarks

The SMSCarrier class contains useful meta-data for each carrier that Simplewire supports.

SMSCarrierList Class

Remarks

The SMSCarrierList class is used internally by the SMS class. A carrier list request sets an internal instance of SMSCarrierList. For access to the contents of that list, getCarrierListIterator() returns an iterator into that list which, when used, must be cast to a SMSCarrier object. See SMSCarrier above.

SMS Methods and Properties

carrierListSend

Syntax

```
sms.carrierListSend();
SMSCarrier carrier;
Iterator I = sms.getCarrierListIterator();
while( I.hasNext() )
{
    carrier = (SMSCarrier) I.next();
    // ...
}
```

Parameters

none

Remarks

carrierListSend(…) makes a request to the Simplewire network for a list of supported carriers. The list contains information for all carriers that Simplewire either supports in their production servers or is developing support for. The SMS object contains an internal SMSCarrierList object. This internal list gets populated when a call to CarrierListSend() has been made. Each SMSCarrier object in the list has useful data concerning the carrier. Users can poll the list for information on any of the carriers. See setOptFields(), getCarrierListIterator().

getCarrierListIterator

Syntax

```
SMSCarrier carrier;
Iterator I = sms.getCarrierListIterator();
while( I.hasNext() )
{
    carrier = (SMSCarrier) I.next();
    // ...
}
```

Parameters

none

Return Value

Iterator for the carrier list object.

Remarks

getCarrierListIterator(…) gets an iterator for the SMS object's internal carrier list object. See carrierListSend(), setOptFields(). See examples.

getConnectionTimeout

Syntax

int i = sms.getConnectionTimeout();

Parameters

none

Return Value

The connection timeout in milliseconds.

Remarks

getConnectionTimeout(...) gets the client-side connection timeout in milliseconds. See setConnectionTimeout().

getErrorCode

Syntax

String s = sms.getErrorCode();

Parameters

none

Remarks

getErrorCode(...) gets the error code returned by the server from the last request. For a complete list of error codes and their corresponding error descriptions download the Simplewire Knowledge Base. See getErrorDesc(), getErrorResolution().

getErrorDesc

Syntax

String s = sms.getErrorDesc();

Parameters

none

Remarks

getErrorDesc(...) gets the error description returned by the server from the last request. For a complete list of error codes and their corresponding error

descriptions, visit the Simplewire Knowledge Base. See getErrorCode(), getErrorResolution().

getErrorResolution

Syntax

String s = sms.getErrorResolution();

Parameters

none

Remarks

getErrorResolution(…) gets the error resolution returned by the server from the last request. For a complete list of error codes and their corresponding error resolutions, visit the Simplewire Knowledge Base. See getErrorCode(), getErrorDesc().

getMsgCallback

Syntax

String s = sms.getMsgCallback();

Parameters

none

Remarks

getMsgCallback(…) gets the message callback value. The message callback is the number that gets dialed when a recipient presses 'talk' on their device after viewing a message. See setMsgCallback().

Note: This feature is not supported on some devices.

getMsgCarrierID

Syntax

String s = sms.getMsgCarrierID();

Parameters

none

Remarks

getMsgCarrierID(…) gets the message carrier ID.

getMsgCLIIconFilename

Syntax

String filename = sms.getMsgCLIIconFilename();

Parameters

none

Remarks

getMsgCLIIconFilename(…) gets the message CLI icon filename. See setMsgCLIIconFilename().

getMsgCLIIconHex

Syntax

String hex = sms.getMsgCLIIconHex();

Parameters

none

Remarks

getMsgCLIIconHex(…) gets the message CLI icon raw hex. See setMsgCLIIconHex().

getMsgFrom

Syntax

String s = sms.getMsgFrom();

Parameters

none

Remarks

getMsgFrom(…) gets the message 'From' value. See setMsgFrom().

getMsgOperatorLogoFilename

Syntax

String filename = sms.getMsgOperatorLogoFilename();

Parameters

none

Remarks

getMsgOperatorLogoFilename(...) gets the message logo filename. See setMsgOperatorLogoFilename().

getMsgOperatorLogoHex

Syntax

String hex = sms.getMsgOperatorLogoHex();

Parameters

none

Remarks

getMsgOperatorLogoHex(...) gets the message logo raw hex. See setMsgOperatorLogoHex().

getMsgPictureFilename

Syntax

String filename = sms.getMsgPictureFilename();

Parameters

none

Remarks

getMsgPictureFilename(...) gets the message picture filename. See setMsgPictureFilename().

getMsgPictureHex

Syntax

String hex = sms.getMsgPictureHex();

Parameters

none

Remarks

getMsgPictureHex(...) gets the message picture raw hex. See setMsgPictureHex().

getMsgPin

Syntax

String s = sms.getMsgPin();

Parameters

none

Remarks

getMsgPin(…) gets the message PIN. See setMsgPin().

getMsgProfileName

Syntax

String profilename = sms.getMsgProfileName();

Parameters

none

Remarks

getMsgProfileName(…) gets the message profile name. See setMsgProfileName().

getMsgProfileRingtone

Syntax

String ringtone = sms.getMsgProfileRingtone();

Parameters

none

Remarks

getMsgProfileRingtone(…) gets the message profile ringtone. See setMsgProfileRingtone().

getMsgProfileScreenSaverFilename

Syntax

String filename = sms.getMsgProfileScreenSaverFilename();

Parameters

none

Remarks

getMsgProfileScreenSaverFilename(...) gets the message profile screensaver filename. See setMsgProfileScreenSaverFilename().

getMsgProfileScreenSaverHex

Syntax

String hex = sms.getMsgProfileScreenSaverHex();

Parameters

none

Remarks

getMsgProfileScreenSaverHex(...) gets the message profile screensaver raw hex. See setMsgProfileScreenSaverHex().

getMsgRingtone

Syntax

String s = sms.getMsgRingtone();

Parameters

none

Remarks

getMsgRingtone(...) gets the message ringtone value. See setMsgRingtone(), setOptPhone().

getMsgStatusCode

Syntax

String s = sms.getMsgStatusCode();

Parameters

none

Remarks

getMsgStatusCode(...) gets the message status code returned by the server. The message status code is the status code of a sent message and is obtained when checking the status of a message. See msgStatusSend().

Note: You must call the method msgStatusSend() first.

getMsgStatusDesc

Syntax

String s = sms.getMsgStatusDesc();

Parameters

none

Remarks

getMsgStatusDesc(...) gets the message status description returned by the server. The message status description is the status description of a sent message and is obtained when checking the status of a message. See msgStatusSend().

Note: You must call the method msgStatusSend() first.

getMsgText

Syntax

String s = sms.getMsgText();

Parameters

none

Remarks

getMsgText(...) gets the message text. See setMsgText().

getMsgTextXML

Syntax

String s = sms.getMsgTextXML;

Parameters

none

Remarks

getMsgTextXML(...) gets the message text XML. See setMsgTextXML().

getMsgTicketID

Syntax

String s = sms.getMsgTicketID();

Parameters
none

Remarks
getMsgTicketID(...) gets the message ticket ID assigned by the server. See setMsgTicketID().

getOptCountryCode

Syntax
String s = sms.getOptCountryCode();

Parameters
none

Remarks
getOptCountryCode(...) gets the option country code. See setOptCountryCode().

getOptDataCoding

Syntax
String s = sms.getOptDataCoding();

Parameters
none

Remarks
getOptDataCoding(...) gets the value for option data coding. See setOptDataCoding().

getOptDelimiter

Syntax
String s = sms.getOptDelimiter();

Parameters
none

Remarks
getOptDelimiter(...) gets the option delimiter. See setOptDelimiter().

getOptFields

Syntax

String s = sms.getOptFields();

Parameters

none

Remarks

getOptFields(...) gets the option fields. See setOptFields().

getOptFlash

Syntax

String s = sms.getOptFlash();

Parameters

none

Remarks

getOptFlash(...) gets the option flash. See setOptFlash().

getOptNetworkCode

Syntax

String s = sms.getOptNetworkCode();

Parameters

none

Remarks

getOptNetworkCode(...) gets the option country code. See setOptNetworkCode().

getOptPhone

Syntax

String optPhone = sms.getOptPhone();

Parameters

none

Remarks

getOptPhone(...) gets the type of phone being sent to. See setOptPhone().

getOptTimeout

Remarks

Deprecated.

getOptType

Syntax

String s = sms.getOptType();

Parameters

none

Remarks

getOptType(...) gets the option type. See setOptType().

getProxyPassword

Syntax

String s = sms.getProxyPassword();

Parameters

none

Remarks

getProxyPassword(...) gets the proxy password. See setProxyPassword().

getProxyPort

Syntax

int i = sms.getProxyPort();

Parameters

none

Remarks

getProxyPort(...) gets the proxy port. See setProxyPort().

getProxyRealm

Syntax

String s = sms.getProxyRealm();

Parameters

none

Remarks

getProxyRealm(...) gets the proxy realm. See setProxyRealm().

getProxyServer

Syntax

String s = sms.getProxyServer();

Parameters

none

Remarks

getProxyServer(...) gets the proxy server. See setProxyServer().

getProxyType

Syntax

int i = sms.getProxyType();

Parameters

none

Remarks

getProxyType(...) gets the proxy type. See getProxyTypeStr(), setProxyType().

getProxyTypeStr

Syntax

String s = sms.getProxyTypeStr();

Parameters

none

Remarks

getProxyTypeStr(...) gets the proxy type in the form of a string.

getProxyUsername

Syntax

String s = sms.getProxyUsername();

Parameters

none

Remarks

getProxyUsername(...) gets the proxy user name. See setProxyUsername().

getServerDomain

Syntax

String s = sms.getServerDomain();

Parameters

none

Remarks

getServerDomain(...) gets the server domain. See setServerDomain().

getServerName

Syntax

String s = sms.getServerName();

Parameters

none

Remarks

getServerName(...) gets the server name. See setServerName().

getServerPort

Syntax

```
int i = sms.getServerPort();
```

Parameters
none

Remarks
getServerPort(...) gets the server port. See setServerPort().

getSubscriberID

Syntax
```
String s = sms.getSubscriberID();
```

Parameters
none

Remarks
getSubscriberID(...) gets the subscriber ID. See setSubscriberID().

getSubscriberPassword

Syntax
```
String s = sms.getSubscriberPassword();
```

Parameters
none

Remarks
getSubscriberPassword(...) gets the subscriber password. See setSubscriberPassword().

getUserAgent

Remarks
Deprecated.

getUserIP

Syntax
```
String s = sms.getUserIP();
```

Parameters

none

Remarks

getUserIP(...) gets the user IP. See setUserIP().

isCarrierList

Syntax

boolean b = sms.isCarrierList();

Parameters

none

Return Value

true, if the last transaction was a carrier list request.

Remarks

isCarrierList(...) returns true if the last transaction was a carrier list request. See carrierListSend().

isMsg

Syntax

boolean b = sms.isMsg();

Parameters

none

Return Value

true, if the last transaction was a send message request.

Remarks

isMsg(...) returns true if the last transaction was a send message request. See msgSend().

isMsgStatus

Syntax

boolean b = sms.isMsgStatus();

Parameters

none

Return Value

true, if the last transaction was a check status request.

Remarks

isMsgStatus(…) returns true if the last transaction was a check status request. See msgStatusSend().

isSuccess

Syntax

boolean b = sms.isSuccess();

Parameters

none

Return Value

true, if the last transaction was successful.

Remarks

isSuccess(…) returns true if the last transaction was successful.

msgSend

Syntax

sms.msgSend();

Parameters

none

Remarks

msgSend(…) sends a message using the Msg properties set up by the user. Once called, the response properties are set and can be polled for error data.

msgSendEx

Syntax

sms.msgSendEx(String carrier, String pin, String from, String callback, String text);

Parameters

carrier – The carrier ID. No longer necessary because of carrier recognition.
pin – The PIN of the mobile device being sent to.
from – The 'from' line.
callback – The callback number.
text – The message text.

Remarks
msgSendEx(…) sends a message on the fly using the parameters provided.

msgStatusSend

Syntax
sms.msgStatusSend();

Parameters
none

Remarks
msgStatusSend(…) checks the status of a page using the ticket ID which is automatically set after a successful msgSend or msgSendEx call. The message ticket ID of a page can also be manually set. See setMsgTicketID(), getMsgStatusCode(), getMsgStatusDesc().

reset

Syntax
sms.reset();

Parameters
none

Remarks
reset(…) resets all SMS properties to their default values.

setConnectionTimeout

Syntax
sms.setConnectionTimeout(int ConnectionTimeout);

Parameters
ConnectionTimeout – the connection timeout in milliseconds

Remarks

setConnectionTimeout(…) sets the client-side connection timeout value in milliseconds. See getConnectionTimeout().

setMsgCallback

Syntax

sms.setMsgCallback(String callback);

Parameters

callback – the callback value

Remarks

setMsgCallback(…) sets the message callback number. The message callback is the number that gets dialed when a recipient presses 'talk' on their device after viewing a message. See getMsgCallback().

Note: This feature is not supported on some devices.

setMsgCarrierID

Syntax

sms.setMsgCarrierID(String carrierID);

Parameters

carrierID – The carrier ID

Remarks

setMsgCarrierID(…) sets the message carrier ID of the recipients wireless device. The message carrier ID is the ID number that Simplewire uses to identify carriers. See getMsgCarrierID(), carrierListSend().

Note: Unless a "345" error code is returned or the carrier is not listed on our carrier list, this property does not need to be set because Simplewire software now performs Global Carrier Recognition.

setMsgCLIIconFilename

Syntax

sms.setMsgCLIIconFilename(String filename);

Parameters

filename – the icon image file

Remarks

setMsgCLIIconFilename(…) sets the message icon filename. In order to be able to send a icon, you must also specify the type of phone being sent to with setOptPhone(). If both text and an image are set, the text will be ignored. Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. The image must be black and white, only. See setMsgCLIIconHex(), setOptPhone().

setMsgCLIIconHex

Syntax

sms.setMsgCLIIconHex(String hex);

Parameters

hex – the icon image raw hex

Remarks

setMsgCLIIconHex(…) sets the message icon raw hex. In order to be able to send a icon, you must also specify the type of phone being sent to with setOptPhone(). If both text and an image are set, the text will be ignored. Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. The image must be black and white, only. See setMsgCLIIconFilename(), setOptPhone().

setMsgFrom

Syntax

sms.setMsgFrom(String from);

Parameters

from – the from value

Remarks

setMsgFrom(…) sets the message name of the sender of the message. The message 'from' text is sent as Unicode. So, the user can use the Java Unicode escape sequence when setting the 'from' text, and the 'from' line will be sent as Unicode. The Java Unicode escape sequence is: Backslash + Lowercase 'u' + Four Hexadecimal Digits.

Example: sms.setMsgFrom("J**\u00FC**rg Freidrich");

This sets the Unicode text "J**ü**rg Freidrich".

Although it is possible to send all Unicode characters in the range of 0x0000 to 0xFFFF in the message, keep in mind that at the time of this writing most mobile

devices *cannot* display Unicode characters beyond 0x00FF. See getMsgFrom(), setOptDataCoding().

setMsgOperatorLogoFilename

Syntax

sms.setMsgOperatorLogoFilename(String filename);

Parameters

filename – the logo image file

Remarks

setMsgOperatorLogoFilename(…) sets the message logo filename. In order to be able to send a logo, you must also specify the type of phone being sent to with setOptPhone(). The country code and network code can also be set for sending a logo. If both text and an image are set, the text will be ignored. Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. The logo must be black and white only, otherwise it will not display correctly.

See setMsgOperatorLogoHex(), setOptPhone(), setOptCountryCode(), setOptNetworkCode().

setMsgOperatorLogoHex

Syntax

sms.setMsgOperatorLogoHex(String hex);

Parameters

hex – the logo image raw hex

Remarks

setMsgOperatorLogoHex(…) sets the message logo raw hex. In order to be able to send a logo, you must also specify the type of phone being sent to with setOptPhone(). The country code and network code can also be set for sending a logo. If both text and an image are set, the text will be ignored. Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. The logo must be black and white only, otherwise it will not display correctly.

See setMsgOperatorLogoFilename(), setOptPhone(), setOptCountryCode(), setOptNetworkCode().

setMsgPictureFilename

Syntax

sms.setMsgPictureFilename(String filename);

Parameters

filename – the picture image file

Remarks

setMsgPictureFilename(…) sets the message picture filename. In order to be able to send a picture, you must also specify the type of phone being sent to with setOptPhone(), and the message text must be set. Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. The image must be black and white, only. See setMsgPictureHex(), setOptPhone(), setMsgText().

setMsgPictureHex

Syntax

sms.setMsgPictureHex(String hex);

Parameters

hex – the picture image raw hex

Remarks

setMsgPictureHex(…) sets the message picture raw hex. In order to be able to send a picture, you must also specify the type of phone being sent to with setOptPhone(), and the message text must be set. Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. The image must be black and white, only. See setMsgPictureFilename(), setOptPhone(), setMsgText().

setMsgPin

Syntax

sms.setMsgPin(String pin);

Parameters

pin – the message pin

Remarks

setMsgPin(…) sets the message pin which is also sometimes referred to as the mobile phone number, MIN, or MSISDN. This is the intended recipient of the message.

International PIN / Mobile Number Format:

```
sms.setMsgPin( "+1 100 510 1234" );
```

The pin is properly set from International use by appending a "+" character followed by the country code and then the national number. See getMsgPin().

setMsgProfileName

Syntax

```
sms.setMsgProfileName( String profilename );
```

Parameters

profilename – the profile name

Remarks

setMsgProfileName(…) sets the message profile name. In order to be able to send a profile, you must also specify the type of phone being sent to with setOptPhone(). See getMsgProfileName(), setMsgProfileRingtone(), setMsgProfileScreenSaverFilename(), setOptPhone().

setMsgProfileRingtone

Syntax

```
sms.setMsgProfileRingtone( String ringtone );
```

Parameters

ringtone – the properly formatted ringtone

Remarks

setMsgRingtone(…) sets the message profile ringtone value. The ringtone must be in RTTTL format. In order to be able to send a profile, you must also specify the type of phone being sent to with setOptPhone(). A profile may consist of any of the following: profile name, profile ringtone, profile screensaver. See getMsgProfileRingtone(), setMsgProfileName(), setMsgProfileScreenSaverFilename(), setOptPhone().

setMsgProfileScreenSaverFilename

Syntax

```
sms.setMsgScreenSaverFilename( String filename );
```

Parameters

filename – the image file

Remarks

setMsgScreenSaverFilename(…) sets the message profile screensaver filename. In order to be able to send a profile, you must also specify the type of phone being sent to with setOptPhone(). A profile may consist of any of the following: profile name, profile ringtone, profile screensaver. Acceptable image formats are: .gif(uncompressed), .jpg , and .jpeg. The image must be black and white, only. See setMsgProfileScreenSaverHex(), setMsgProfileName(), setMsgProfileRingtone(), setOptPhone().

setMsgProfileScreenSaverHex

Syntax

sms.setMsgScreenSaverHex(String hex);

Parameters

hex – the screen saver image raw hex

Remarks

setMsgScreenSaverHex(…) sets the message profile screensaver raw hex. In order to be able to send a profile, you must also specify the type of phone being sent to with setOptPhone(). A profile may consist of any of the following: profile name, profile ringtone, profile screensaver. Acceptable image formats are: .gif(uncompressed), .jpg , and .jpeg. The image must be black and white, only. See setMsgProfileScreenSaverFilename(), setMsgProfileName(), setMsgProfileRingtone(), setOptPhone().

setMsgRingtone

Syntax

sms.setMsgRingtone(String ringtone);

Parameters

ringtone – the properly formatted ringtone

Remarks

setMsgRingtone(…) sets the message ringtone value. The ringtone must be in RTTTL format. In order to be able to send a ringtone, you must also specify the type of phone being sent to with setOptPhone(). If both text and a ringtone are set, the text will be ignored. See getMsgRingtone(), setOptPhone().

setMsgText

Syntax

sms.setMsgText(String text);

Parameters

text – the message text

Remarks

setMsgText(…) sets the message text. The message text is sent as Unicode. So, the user can use the Java Unicode escape sequence when setting the message text and the message will be sent as Unicode. The Java Unicode escape sequence is: Backslash + Lowercase 'u' + Four Hexadecimal Digits.

Example: sms.setMsgText("Smiley Face: \u263A");

This sets the Unicode text "Smiley Face: ☺".

Although it is possible to send all Unicode characters in the range of 0x0000 to 0xFFFF in the message, keep in mind that at the time of this writing most mobile devices *cannot* display Unicode characters beyond 0x00FF. See getMsgText(), setOptDataCoding().

setMsgTextXML

Syntax

sms.setMsgTextXML(String textXML);

Parameters

textXML – the message text XML

Remarks

setMsgTextXML(…) sets the message text XML. The message text XML is sent as 7-bit data. The user can set the message text to raw XML. The XML document string passed in must contain only 7-bit characters. The XML passed in must be a properly formatted XML document. See getMsgTextXML().

setMsgTicketID

Syntax

sms.setMsgTicketID(String ticketID);

Parameters

ticketID – ticket ID

Remarks

setMsgTicketID(…) sets the message ticket ID. The ticket ID is the handle to a message sent back from the Simplewire servers when the message is sent. The only purpose in manually setting the ticket ID would be to check the status of a message with a given ID. See getMsgTicketID(), msgStatusSend().

setOptCountryCode

Syntax

sms.setOptCountryCode(String optCountryCode);

Parameters

optCountryCode – the option country code

Remarks

setOptCountryCode(…) sets the option country code. This option can be set for sending operator logos. See getOptCountryCode().

setOptDataCoding

Syntax

sms.setOptDataCoding(String optDataCoding);

Parameters

optDataCoding – the data coding scheme

Remarks

setOptDataCoding(…) sets the option that tells the server how to encode the text being sent to the mobile device. Only newer mobile devices accept 8-bit or higher text. Most older devices can only understand 7-bit encoding. The values accepted are “AUTO”, “7BIT”, “8BIT”, and “UCS2”. Encoding using 7 bits can represent Unicode characters in the range 0x0000 to 0x007F. Encoding using 8 bits can represent Unicode characters in the range 0x0000 to 0x00FF. Encoding using UCS2 can represent Unicode characters in the range 0x0000 to 0xFFFF. If the wrong setting is used (e.g. “8BIT” when “7BIT” is required), the text may not display correctly on the recipient’s mobile device. If no value is specified, the server assumes “AUTO”. See getOptDataCoding().

setOptDelimiter

Syntax

sms.setOptDelimiter(String optDelimiter);

Parameters

optDelimiter – the option delimiter

Remarks

setOptDelimiter(…) sets the option delimiter. The option delimiter is the string that separates the different fields when a message gets formatted. By default, this property is not defined, and therefore not used. See getOptDelimiter().

setOptFields

Syntax

sms.setOptFields(String optFields);

Parameters

optFields – the option fields specifier

Remarks

setOptFields(…) sets the string which specifies the return fields for a carrier list request. Accepted values are “all” for the meta-data (e.g. callback supported, max text length, etc.), or “selectbox” for just the essential carrier data (i.e. carrier title, subtitle, and ID). See getOptFields().

setOptFlash

Syntax

sms.setOptFlash(String optFlash);

Parameters

optFlash – the option flash (i.e. “true”)

Remarks

SetOptFlash(…) is the string which specifies whether the message should flash (not blink) on the recipient’s device rather than being saved. Sometimes this is called a “Newsflash” message. Accepted values are “true”. This feature is not supported on all devices and it will be ignored on unsupported devices and carriers.

setOptNetworkCode

Syntax

sms.setOptNetworkCode(String optNetworkCode);

Parameters

optNetworkCode – the option network code

Remarks

setOptNetworkCode(…) sets the option network code. This option can be set for sending operator logos. See getOptNetworkCode().

setOptPhone

Syntax

sms.setOptPhone(String optPhone);

Parameters

optPhone – the phone type

Remarks

setOptPhone(…) sets the type of phone being sent to. For example, sms.setOptPhone(“Nokia”). If you are sending either a ringtone or a logo, you must specify the phone type. In version 2.4.0, only Nokia phones are supported. In the future we will support Motorola, Ericsson, etc. See getOptPhone().

setOptTimeout

Remarks

Deprecated.

setOptType

Syntax

sms.setOptType(String optType);

Parameters

optType – option type

Remarks

setOptType(…) sets the type of carrier list that should be returned. Accepted values are:

“production” - returns the current carriers that Simplewire supports in their production servers

“development” - returns the carriers that Simplewire is developing support for

See getOptType().

setProxyPassword

Syntax

sms.setProxyPassword(String proxyPassword);

Parameters

proxyPassword – the proxy password

Remarks

setProxyPassword(…) sets the password that goes along with the proxy username. A valid username and password are necessary in order to use a proxy server which requires authentication. See getProxyPassword().

setProxyPort

Syntax

sms.setProxyPort(int proxyPort);

Parameters

proxyPort – the proxy port to connect to

Remarks

setProxyPort(…) sets the port number of the proxy server to use for users behind proxy firewalls. Often, this value is 1080. See getProxyPort().

setProxyRealm

Syntax

sms.setProxyRealm(String proxyRealm);

Parameters

proxyRealm – the proxy realm name

Remarks

setProxyRealm(…) sets the proxy server realm. This property must be set when using HTTP Proxy. The realm name must be exact, so beware of any prefixes or suffixes. For example, a server with the name “sincic”, might have a realm name of “`sincic.`” or “`@sincic`” depending on how the server is setup. Some servers have prefixes or suffixes in the realm name for simplified authentication, others do

not. If you don't know the exact proxy server realm, contact your network administrator. See getProxyRealm().

setProxyServer

Syntax

sms.setProxyServer(String proxyServer);

Parameters

proxyServer – the proxy server name

Remarks

setProxyServer(…) sets the proxy server name. A port must be specified with setProxyPort(). For HTTP Proxy, the server realm must be set with setProxyRealm(). A type of proxy must be specified with setProxyType(). See getProxyServer().

setProxyType

Syntax

sms.setProxyType(int proxyType);

Parameters

proxyType – a class-constant specifying the type of proxy

Remarks

setProxyType(…) sets the type of proxy connection – HTTP, SOCKS4, or SOCKS5. Use the class-constants SMS.PROXY_TYPE_HTTP, SMS.PROXY_TYPE_SOCKS4, and SMS.PROXY_TYPE_SOCKS5 to set the proxy type. See getProxyType().

setProxyUsername

Syntax

sms.setProxyUsername(String proxyUsername);

Parameters

proxyUsername – the user name to use for proxy authentication

Remarks

setProxyUsername(…) sets the user name to use for proxy authentication. Usually, a proxy password must be set as well. See getProxyUsername().

setServerDomain

Syntax

sms.setServerDomain(String serverDomain);

Parameters

serverDomain – the server domain

Remarks

setServerDomain(…) sets the server domain to use for the connection. The server domain works in conjunction with the server name to produce the URL to which the current message gets posted. This value is preset and should not need to be changed. See getServerDomain().

setServerName

Syntax

sms.setServerName(String serverName);

Parameters

serverName – the name of the server

Remarks

setServerName(…) sets the name of the server for use in the connection. The server name works in conjunction with the server domain to produce the URL to which the current message gets posted. This value is preset and should not need to be changed, unless you are a Beta-Tester. See getServerName().

setServerPort

Syntax

sms.setServerPort(int serverPort);

Parameters

serverPort – the server port

Remarks

setServerPort(…) sets the port to which the SDK connects on the server. The default port is 80. If you are a Beta-Tester, you may need to change this value. See getServerPort().

setSubscriberID

Syntax

sms.setSubscriberID(String subscriberID);

Parameters

subscriberID – the Simplewire-provided subscriber ID

Remarks

setSubscriberID(…) sets the ID of a subscriber. The subscriber ID is an ID number provided to paid subscribers that gives access to all of Simplewire's resources. The appropriate password must also be set. See getSubscriberID().

setSubscriberPassword

Syntax

sms.setSubscriberPassword(String subscriberPassword);

Parameters

subscriberPassword – the password that goes along with the subscriber ID

Remarks

setSubscriberPassword(…) sets the password that goes along with the subscriber ID. Each paid subscriber is given a unique ID. Each subscriber ID has an associated password. Both the ID and the password must be set correctly. See getSubscriberPassword().

setUserAgent

Remarks

Deprecated.

setUserIP

Syntax

sms.setUserIP(String userIP);

Parameters

userIP – IP of the user

Remarks

setUserIP(…) sets the IP address of the sender. See getUserIP().

xmlParse

Syntax

sms.xmlParse();

Parameters

none

Remarks

xmlParse(...) parses the XML currently held by the SMS object, and sets the properties.

xmlParseEx

Syntax

sms.xmlParseEx(String xml);

Parameters

xml – an XML string

Remarks

xmlParseEx(...) calls setXML() with the XML passed in and then calls xmlParse().

SMSCarrier Properties

The SMSCarrier object contains useful meta-data for each carrier that Simplewire supports. Example meta-data includes, but is not limited to, maximum text message length, callback support, 'from' requirement. Each carrier has a carrier id associated with it, which the Simplewire network uses to connect to the carrier networks to send the request. When the Option Fields property is set to "all", each property below is returned. When the Option Fields property is set to "selectbox" only the id, title, and subtitle are returned.

Carrier ID

Syntax

```
String s = carrier.getCarrierID();
```

Remarks

The ID which is used by Simplewire to identify carriers.

Title

Syntax

```
String s = carrier.getTitle();
```

Remarks

The title of the carrier.

Subtitle

Syntax

```
String s = carrier.getSubtitle();
```

Remarks

The subtitle of the carrier.

Content Type

Syntax

```
String s = carrier.getContentType();
```

Remarks

The content type of the carrier.

Carrier Country Code

Syntax

String s = carrier.getCountryCode();

Remarks

The country code of the country that the carrier resides in.

Country Name

Syntax

String s = carrier.getCountryName();

Remarks

The country that the carrier is in.

Country Region

Syntax

String s = carrier.getCountryRegion();

Remarks

The country region.

Pin Required

Syntax

String s = carrier.isPinRequired();

Remarks

Indicates if the PIN is required.

Pin Minimum Length

Syntax

String s = carrier.getPinMinLength();

Remarks

The minimum length of the PIN.

Pin Maximum Length

Syntax

String s = carrier.getPinMaxLength();

Remarks

The maximum length of the PIN.

Text Required

Syntax

String s = carrier.isTextRequired();

Remarks

Indicates if text is required.

Text Minimum Length

Syntax

String s = carrier.getTextMinLength();

Remarks

The minimum length of the text.

Text Maximum Length

Syntax

String s = carrier.getTextMaxLength();

Remarks

The maximum length of the text.

From Required

Syntax

String s = carrier.isFromRequired();

Remarks

Indicates if the from line is required.

From Minimum Length

Syntax

String s = carrier.getFromMinLength();

Remarks

The minimum length of the from line.

From Maximum Length

Syntax

String s = carrier.getFromMaxLength();

Remarks

The maximum length of the from line.

Callback Required

Syntax

String s = carrier.isCallbackRequired();

Remarks

Indicates if callback is required.

Callback Supported

Syntax

String s = carrier.isCallbackSupported();

Remarks

Indicates if callback is supported.

Callback Minimum Length

Syntax

String s = carrier.getCallbackMinLength();

Remarks

The minimum length of the callback.

Callback Maximum Length

Syntax

String s = carrier.getCallbackMaxLength();

Remarks

The maximum length of the callback.

Type

Syntax

String s = carrier.getType();

Remarks

The type.

Smart Messaging Supported

Syntax

String s = carrier.getSmartMsgID();

Remarks

Smart Message support.

SMSCarrierList

SMSCarrierList is an automated collection of SMSCarrier objects. The SMSCarrierList is not directly used by the user. An internal SMSCarrierList object is contained in each SMS object. The internal SMSCarrierList object is populated after a successful carrierListSend(). The user can call the getCarrierListIterator() method to obtain an iterator to the internal SMSCarrier object, which can be used to access the data in the SMSCarrierList object.