Perl SMS SDK 2.4.0 Documentation

Abstract

This document describes the methods and properties of the Simplewire Perl SMS SDK and gives example code demonstrating how to implement Simplewire-aware software.

Table of Contents

Introduction 4

UNIX Install Instructions 5

Windows Install Instructions 7

Objects Overview 9
SMS Object 9
Carrier Object 9
Carrier List 9

SMS Methods and Properties 10
carrierList 10
carrierListSend 10
connectionTimeout 11
errorCode 11
errorDesc 11
isCarrierList 12
isMsg 12
isMsgStatus 12
msgCallback 13
msgCarrierID 13
msgCLIIconFilename 14
msgFrom 14
msgOperatorLogoFilename 15
msgPictureFilename 15
msgPin 16
msgProfileName 16
msgProfileRingtone 16
msgProfileScreenSaverFilename 17
msgRingtone 17
msgSend 17
msgSendEx 18
msgStatusCode 18
msgStatusDesc 19
msgStatusSend 19
msgText 19
msgTicketID 20
optCountryCode 21
optDataCoding 21
optDelimiter 22
optFields 22
optNetworkCode 23
optPhone 23
optTimeout 23

optType 23
proxyPassword 24
proxyPort 24
proxyServer 25
proxyUsername 25
Reset 25
serverDomain 26
serverName 26
serverPort 27
subscriberID 27
subscriberPassword 27
success 28
userAgent 28
userIP 28
toXML 29
xmlParse 29
xmlParseEx 29

Carrier Object Properties 31
Carrier ID 31
Title 31
Subtitle 31
Content Type 32
Country Code 32
Country Name 32
Country Region 32
Pin Required 33
Pin Minimum Length 33
Pin Maximum Length 33
Text Required 34
Text Minimum Length 34
Text Maximum Length 34
From Required 35
From Minimum Length 35
From Maximum Length 35
Callback Required 35
Callback Supported 36
Callback Minimum Length 36
Callback Maximum Length 36
Type 37
Smart Message Support 37

Carrier List 38

Examples 39

Sending a SMS Message in Perl 39
Retrieving a Carrier List in Perl 40
Checking Message Status in Perl 41

Support 43

Disclaimer 44

Introduction

The Perl SMS SDK provides easy, high-level control of the Simplewire wireless messaging platform. The Perl 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 Perl SMS SDK makes it possible to send a wireless message with as little as two lines of code.

The Perl SMS SDK provides an enterprise level implementation. You can use it on any platform with Perl installed on it. The SDK has gone through rigorous testing to ensure a comphrehensive, best-of-breed SMS SDK. In turn, the Perl SMS SDK delivers a robust business solution for high-performance, web based applications in a variety of industries.

UNIX Install Instructions

The installation of the Perl SMS SDK requires the following steps.

1. Download the correct .tar.gz installation package. If your platform is not supported then please send Simplewire an email to support@simplewire.com so that we can commercially support it. Otherwise, follow the steps below.

2. Unzip and untar the installation package.

   [root]% tar –zxvf Net-SMS-X.XX.tar.gz

3. Change directories to the Net-SMS-X.XX directory

   [root]% cd Net-SMS-X.XX

4. The Perl SMS SDK has some prerequisite modules. These can be installed manually, or using the CPAN utility.

   ```
   [Net-SMS-X.XX]% CPAN
   cpan> install HTTP::Request
   cpan> install HTTP::Response
   cpan> install LWP::UserAgent
   cpan> install Unicode::String
   cpan> install XML::DOM
   cpan> q
   ```

5. Create the makefile

   [Net-SMS-X.XX]% perl Makefile.PL

6. Run the 'make' command

   [Net-SMS-X.XX]% make

7. Run "make install"

   [Net-SMS-X.XX]% make install

8. Test the installation by running an example.

   [Net-SMS-X.XX]% cd examples

   [examples]% perl send_text.pl

Example code for using the SDK can be found in: \Net-SMS-X.XX\examples.

The Perl SMS SDK has the following prerequisites:

Unicode::String	2.06+
XML::DOM	1.25+
LWP::UserAgent	(Part of libwww-perl 5.5394+)
HTTP::Request	(Part of libwww-perl 5.5394+)
HTTP::Response	(Part of libwww-perl 5.5394+)

Windows Install Instructions

The installation of the Perl SMS SDK requires the following steps. To use Perl on Windows you should use an up-to-date version of ActiveState's ActivePerl available at www.activestate.com. The Perl SMS SDK was tested using ActivePerl 5.6.1-Build-629. You will also need the NMAKE utility from Microsoft which can be found at ftp://ftp.microsoft.com/Softlib/mslfiles/nmake15.exe

1. Download the correct .tar.gz installation package. If your platform is not supported then please send Simplewire an email to support@simplewire.com so that we can commercially support it. Otherwise, follow the steps below.

2. Unzip and untar the installation package.

3. Change directories to the Net-SMS-X.XX directory

   > cd Net-SMS-X.XX

4. The Perl SMS SDK has some prerequisite modules. Most of these are included with ActivePerl. A few are not. You can install these with the Perl Package Manager (PPM) included with ActivePerl, or you can run a batch command included with the Perl SMS SDK release.

   > InstallModules

   -- OR --

   > PPM
   PPM interactive shell (X.X.X)
   PPM> install Unicode-String
   PPM> install XML-DOM
   PPM> quit

5. Create the makefile

   > perl Makefile.PL

6. Run the 'nmake' command

   > nmake

7. Run "nmake install"

   > nmake install

8. Test the installation by running an example from the example folder: \Net-SMS-X.XX\examples\

   > cd examples

   > perl send_text.pl

The Perl SMS SDK has the following prerequisites:

Unicode::String	2.06+
XML::DOM	1.25+
LWP::UserAgent	(Part of libwww-perl 5.5394+)
HTTP::Request	(Part of libwww-perl 5.5394+)
HTTP::Response	(Part of libwww-perl 5.5394+)

Objects Overview

There is one object module in the SDK package, the SMS module, and two Perl object types, the carrier object and the carrier list.

SMS Object

Syntax

```
my $sms = Net::SMS->new();
```

Remarks

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

Carrier Object

Syntax

```
$sms = Net::SMS->new();
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{ID} . " " . $row->{Title} . " " . $row->{SubTitle} . "\n";
}
```

Remarks

The Carrier object is a hash containing useful meta-data for each carrier that Simplewire supports.

Carrier List

Syntax

```
$sms = Net::SMS->new();
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{ID} . " " . $row->{Title} . " " . $row->{SubTitle} . "\n";
}
```

Remarks

The Carrier List is an array of Carrier objects. A successful carrier list request populates an internal array. This array can be returned to the user.

SMS Methods and Properties

carrierList

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{ID} . " " . $row->{Title} . " " . $row->{SubTitle} . "\n";
}
```

Parameters

none

Return Value

Array of carrier information.

Remarks (read only)

carrierList(…) returns an array that contains information for all the carriers that Simplewire either supports in their production servers or is developing support for. The SMS object contains an internal carrier array. This internal array gets populated when a call to carrierListSend() has been made. Each carrier object in the array has useful data concerning the carrier. Users can poll the array for information on any of the carriers. See optFields(), carrierListSend().

carrierListSend

Syntax

```
$sms->carrierListSend();
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{ID} . " " . $row->{Title} . " " . $row->{SubTitle} . "\n";
}
```

Parameters

none

Return Value

1, if successful request. 0, otherwise.

Remarks (read only)

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 carrier array object. This internal list gets populated

when a call to carrierListSend() has been made. Each carrier object in the array has useful data concerning the carrier. Users can poll the list for information on any of the carriers. See optFields(), carrierList().

connectionTimeout

Syntax

```
$timeout = $sms->connectionTimeout();
$sms->connectionTimeout( $timeout );
```

Parameters

The connection timeout value in seconds.

Return Value

The connection timeout value in iseconds.

Remarks (read/write)

connectionTimeout(…) gets/sets the client-side connection timeout in seconds.

errorCode

Syntax

```
$code = $sms->errorCode();
```

Parameters

none

Return Value

The error code.

Remarks (read only)

errorCode(…) 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 errorDesc().

errorDesc

Syntax

```
$desc = $sms->errorDesc();
```

Parameters

none

Remarks

errorDesc(…) gets the error description 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 errorCode().

isCarrierList

Syntax

$boolean = $sms->isCarrierList();

Parameters

none

Return Value

1, if the last transaction was a carrier list request. 0, otherwise.

Remarks (read only)

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

isMsg

Syntax

$boolean = $sms->isMsg();

Parameters

none

Return Value

1, if the last transaction was a send message request. 0, otherwise.

Remarks (read only)

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

isMsgStatus

Syntax

$boolean = $sms->isMsgStatus();

Parameters

none

Return Value
1, if the last transaction was a check status request. 0, otherwise.

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

msgCallback

Syntax
$callback = $sms->msgCallback();
$sms->msgCallback($callback);

Parameters
callback – the callback value

Return Value
The callback value.

Remarks
msgCallback(…) gets/sets 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.

Note: This feature is not supported on some devices.

msgCarrierID

Syntax
$carrierID = $sms->msgCarrierID();
$sms->msgCarrierID($carrierID);

Parameters
carrierID – The carrier ID

Return Value
The carrier ID.

Remarks (read/write)

msgCarrierID(…) gets/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 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.

msgCLIIconFilename

Syntax

$filename = $sms->msgCLIIconFilename();

$sms->msgCLIIconFilename("myImageFile.gif");

Remarks (read/write)

msgCLIIconFilename(…) is the filename of an image file to be used as the message icon value. In order to send an icon you must also specify the type of phone being sent to with optPhone(). Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. Image must be black and white, only. See optPhone().

msgFrom

Syntax

$from = $sms->msgFrom();

$sms->msgFrom($from);

Parameters

from – the from value

Return Value

The from value.

Remarks (read/write)

msgFrom(…) gets/sets the message name of the sender of the message. The message 'from' text is sent as Unicode. The user can use one of two methods to send Unicode characters. Both of the following methods are for use with double-quoted strings only, not single-quoted strings. Versions of Perl 5.6 and above use UTF-8 encoding internally, and provide a hexadecimal escape sequence:

Syntax: Backslash + Lowercase 'x' + Two Hexadecimal Digits

Users can use this escape sequence when setting the 'from' text, and the 'from' line will be sent as Unicode.

Example: $sms->msgFrom("J**\xFC**rg Freidrich");

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

The Perl hexadecimal escape allows users to enter Unicode characters in the limited range 0x0000 to 0x00FF. Simplewire provides its own escape sequence:

Syntax: Backslash + Backslash + Uppercase 'X' + Four Hexadecimal Digits.

The Simplewire escape can be used to enter Unicode characters in the full range 0x0000 to 0xFFFF.

Example: $sms->msgFrom("J**\\X00FC**rg Freidrich");

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

But, with the Simplewire escape you can use a wider range of Unicode characters.

Example: $sms->msgFrom("MySmileyCompany **\\X263A**");

This example sets the Unicode text "MySmileyCompany ☺".

See optDataCoding().

Note: 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.

msgOperatorLogoFilename

Syntax

$sms->msgOperatorFilename("myImageFile.gif");

Remarks (read/write)

msgOperatorLogoFilename(…) is the filename of an image file to be used as the message logo value. In order to send a logo you must also specify the type of phone being sent to with optPhone(). The country code and network code should also be set when sending a logo. If both text and an image are set, the text will be ignored. The image must be no larger than 72x14 pixels. Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. Image must be black and white, only. See optPhone(), optCountryCode(), optNetworkCode().

msgPictureFilename

Syntax

$sms->msgPictureFilename("myImageFile.gif");

Remarks (read/write)

msgPictureFilename(…) is the filename of an image file to be used as the message picture value. In order to send a picture you must also specify the type of phone being sent to with optPhone(), and the message text should also be set. The image must be no larger than 72x28 pixels. Acceptable picture formats are:

.gif(uncompressed), .jpg , and .jpeg. Image must be black and white, only. See optPhone(), msgText().

msgPin

Syntax

```
$pin = $sms->msgPin();
$sms->msgPin( $pin );
```

Parameters

pin – the message pin

Return Value

The message pin.

Remarks (read/write)

msgPin(…) gets/sets the message PIN. The PIN is the pager/phone ID number. The PIN is typically the phone number of the device or the MIN (Mobile Identification Number). There are two formats for the PIN.

United States PIN / Mobile Number Format:

```
$sms->msgPin( "1005101234" );
```

International PIN / Mobile Number Format:

```
$sms->msgPin( "+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.

msgProfileName

Syntax

```
$sms->msgProfileName( "myProfileName" );
```

Remarks (read/write)

msgProfileName(…) is the message profile name. A profile can consist of any of the following: profile name, profile ringtone, profile screensaver. In order to send a profile you must also specify the type of phone being sent to with optPhone(). See msgProfileRingtone(), msgProfileScreenSaverFilename(), optPhone().

msgProfileRingtone

Syntax

$sms->msgProfileRingtone("RTTTL ringtone");

Remarks (read/write)

msgProfileRingtone(…) is the ringtone part of a message profile. A profile can consist of any of the following: profile name, profile ringtone, profile screensaver. The ringtone must be in RTTTL format. In order to send an profile you must also specify the type of phone being sent to with optPhone(). See msgProfileName(), msgProfileScreenSaverFilename(), optPhone().

msgProfileScreenSaverFilename

Syntax

$sms->msgProfileScreenSaverFilename("myImageFile.gif");

Remarks (read/write)

msgProfileScreenSaverFilename(…) is the filename of an image file to be used as the message profile screensaver value. A profile can consist of any of the following: profile name, profile ringtone, profile screensaver. In order to send a profile you must also specify the type of phone being sent to with optPhone(). The image must be no larger than 72x28 pixels. Acceptable picture formats are: .gif(uncompressed), .jpg , and .jpeg. Image must be black and white, only. See msgProfileName(), msgProfileRingtone(), optPhone().

msgRingtone

Syntax

$sms->msgRingtone("RTTTL ringtone");

Remarks (read/write)

msgRingtone(…) is the ringtone to be sent as a message. The ringtone must be in RTTTL format. In order to send an ringtone you must also specify the type of phone being sent to with optPhone(). See optPhone().

msgSend

Syntax

$sms->msgSend();

Parameters

none

Return Value

1 for successful request. 0, otherwise.

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.
pin – The PIN of the mobile device being sent to.
from – The 'from' line.
callback – The callback number.
text – The message text.

Return Value

1 for successful request. 0, otherwise.

Remarks

msgSendEx(…) sends a message on the fly using the parameters provided. See msgSend(), msgCarrierID(), msgPin(), msgFrom(), msgCallback(), msgText().

msgStatusCode

Syntax

$code = $sms->msgStatusCode();

Parameters

none

Return Value

The message status code.

Remarks (read only)

msgStatusCode(…) 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 msgSendStatus(), msgStatusDesc().

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

msgStatusDesc

Syntax
$desc = $sms->msgStatusDesc();

Parameters
none

Return Value
The message status description.

Remarks (read only)
msgStatusDesc(…) 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 msgSendStatus(), msgStatusCode().

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

msgStatusSend

Syntax
$sms->msgStatusSend();

Parameters
none

Return Value
1, for successful request. 0, otherwise.

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 msgTicketID(), msgStatusCode(), msgStatusDesc().

msgText

Syntax
$text = $sms->msgText();

$sms->msgText($text);

Parameters
text – the message text

Return Value
The text message.

Remarks (read/write)
msgText(…) gets/sets the message text. The message text is sent as Unicode. The user can use one of two methods to send Unicode characters. Both of the following methods are for use with double-quoted strings only, not single-quoted strings. Versions of Perl 5.6 and above use UTF-8 encoding internally, and provide a hexadecimal escape sequence:

Syntax: Backslash + Lowercase 'x' + Two Hexadecimal Digits

Users can use this escape sequence when setting the message text, and the message text will be sent as Unicode.

Example: $sms->msgText("Your stock is up: +9**\xBC**");

This example sets the Unicode text "Your stock is up: +**9** ¼ ".

The Perl hexadecimal escape allows users to enter Unicode characters in the limited range 0x0000 to 0x00FF. Simplewire provides its own escape sequence:

Syntax: Backslash + Backslash + Uppercase 'X' + Four Hexadecimal Digits.

The Simplewire escape can be used to enter Unicode characters in the full range 0x0000 to 0xFFFF.

Previous Example: $sms->msgText("Your stock is up: +9**\\X00BC**");

This example sets the Unicode text "Your stock is up: +**9** ¼ ".

But, with the Simplewire escape you can use a wider range of Unicode characters.

Example: $sms->msgFrom("Hello There! **\\X263A**");

This example sets the Unicode text "Hello There! ☺".

See optDataCoding().

Note: 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.

msgTicketID

Syntax
$ticketID = $sms->msgTicketID();
$sms->msgTicketID($ticketID);

Parameters
ticketID – ticket ID

Return Value
The ticket ID.

Remarks (read/write)
msgTicketID(…) gets/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 msgStatusSend().

optCountryCode

Syntax
$countrycode = $sms->optCountryCode();
$sms->optCountryCode($countrycode);

Parameters
countrycode – the country code of the recipient's service provider

Return Value
The country code.

Remarks (read/write)
optCountryCode(…) gets/sets the country code of the carrier who provides service to the mobile device being sent to. This value should be set when sending an operator logo. See msgOperatorLogo().

optDataCoding

Syntax
$datacoding = $sms->optDataCoding();
$sms->optDataCoding($datacoding);

Parameters
datacoding – the data coding scheme

Return Value
The data coding scheme.

Remarks (read/write)
optDataCoding(…) gets/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 msgFrom(), msgText().

optDelimiter

Syntax

```
$delimiter = $sms->optDelimiter();
$sms->optDelimiter( $delimiter );
```

Parameters

delimiter – the option delimiter

Return Value

The option delimiter.

Remarks (read/write)

optDelimiter(…) gets/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.

optFields

Syntax

```
$fields = $sms->optFields();
$sms->optFields( $fields );
```

Parameters

fields – the option fields specifier

Return Value

The option fields specifier.

Remarks (read/write)

optFields(…) gets/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 carrierListSend().

optNetworkCode

Syntax

$networkcode = $sms->optNetworkCode();

$sms->optNetworkCode($networkcode);

Parameters

networkcode – the network code of the recipient's service provider

Return Value

The network code.

Remarks (read/write)

optNetworkCode(…) gets/sets the network code of the carrier who provides service to the mobile device being sent to. This value should be set when sending an operator logo. See msgOperatorLogo().

optPhone

Syntax

$phone = $sms->optPhone();

$sms->optPhone($phone);

Parameters

phone – the type of the recipient's phone

Return Value

The phone type.

Remarks (read/write)

optPhone(…) gets/sets the type of phone being sent to. This property must be set when sending a ringtone, logo, icon, picture, or profile. In version 2.4.0, only Nokia phones are supported. In the future we will support Motorola, Ericsson, etc.

optTimeout

Remarks

Deprecated.

optType

Syntax

```
$type = $sms->optType();
$sms->optType( $type );
```

Parameters

type – option type

Return Value

The option type.

Remarks (read/write)

optType(…) gets/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 carrierListSend().

proxyPassword

Syntax

```
$password = $sms->proxyPassword();
$sms->proxyPassword( $password );
```

Parameters

password – the proxy password

Return Value

The proxy password.

Remarks (read/write)

proxyPassword(…) gets/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 proxyUsername(), proxyServer().

proxyPort

Syntax

```
$port = $sms->proxyPort();
$sms->proxyPort( $port );
```

Parameters

port – the proxy port to connect to

Return Value

The proxy port.

Remarks (read/write)

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

proxyServer

Syntax

```
$server = $sms->proxyServer();
$sms->proxyServer( $server );
```

Parameters

server – the proxy server name

Return Value

The proxy server name.

Remarks (read/write)

proxyServer(…) gets/sets the proxy server name. A port must be specified with proxyPort().

proxyUsername

Syntax

```
$username = $sms->proxyUsername();
$sms->proxyUsername( $username);
```

Parameters

username – the user name to use for proxy authentication

Return Value

The user name to use for proxy authentication.

Remarks (read/write)

proxyUsername(…) gets/sets the user name to use for proxy authentication. Usually, a proxy password must be set as well. See proxyPassword(), proxyServer().

Reset

Syntax

$sms->Reset();

Parameters

none

Remarks

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

serverDomain

Syntax

$domain = $sms->serverDomain();
$sms->serverDomain($domain);

Parameters

domain – the server domain

Return Value

The server domain.

Remarks (read/write)

serverDomain(…) gets/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, unless you are a Beta-Tester.

serverName

Syntax

$name = $sms->serverName();
$sms->serverName($name);

Parameters

name – the name of the server

Return Value

The name of the server.

Remarks (read/write)

serverName(…) gets/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.

serverPort

Syntax

```
$port = $sms->serverPort();
$sms->serverPort( $port );
```

Parameters

port – the server port

Return Value

The server port.

Remarks (read/write)

serverPort(…) gets/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.

subscriberID

Syntax

```
$ID = $sms->subscriberID();
$sms->subscriberID( $ID );
```

Parameters

ID – the Simplewire-provided subscriber ID

Return Value

The Simplewire-provided subscriber ID.

Remarks (read/write)

subscriberID(…) gets/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 subscriberPassword().

subscriberPassword

Syntax

```
$password = $sms->subscriberPassword();
$sms->subscriberPassword( $password );
```

Parameters

password – the password that goes along with the subscriber ID

Return Value

The subscriber password.

Remarks (read/write)

subscriberPassword(…) gets/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 subscriberID().

success

Syntax

```
$boolean = $sms->success();
```

Parameters

none

Return Value

1, if the last transaction was successful. 0, otherwise.

Remarks (read only)

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

userAgent

Remarks

Deprecated.

userIP

Syntax

```
$IP = $sms->userIP();
$sms->userIP( $IP );
```

Parameters

IP – IP of the user

Return Value

IP of the user.

Remarks (read/write)

userIP(…) sets the IP address of the sender.

toXML

Syntax

$xml_str = $sms->toXML();

Parameters

none

Return Value

The currently stored XML.

Remarks (read only)

toXML(…) forms an XML request using the current properties. This XML request is stored internally, and also returned as a string. The currently stored XML can be parsed by calling xmlParse().

xmlParse

Syntax

$sms->xmlParse();

Parameters

none

Return Value

none

Remarks

xmlParse(…) calls toXML() to form the XML request and passes the result into xmlParseEx().

xmlParseEx

Syntax

$sms->xmlParseEx($xml_str);

Parameters

xml_str – an XML string

Remarks

xmlParseEx(…) sets the currently stored XML to the string passed in, then parses it and sets properties.

Carrier Object Properties

The Carrier 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

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{ID} . "\n";
}
```

Remarks

The ID which is used by Simplewire to identify carriers.

Title

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{Title} . "\n";
}
```

Remarks

The title of the carrier.

Subtitle

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{SubTitle} . "\n";
}
```

Remarks

The subtitle of the carrier.

Content Type

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{ContentType} . "\n";
}
```

Remarks

The content type of the carrier.

Country Code

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{CountryCode} . "\n";
}
```

Remarks

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

Country Name

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{CountryName} . "\n";
}
```

Remarks

The country that the carrier is in.

Country Region

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{CountryRegion} . "\n";
}
```

Remarks

The country region.

Pin Required

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{PinRequired} . "\n";
}
```

Remarks

Indicates if the PIN is required.

Pin Minimum Length

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{PinMinLength} . "\n";
}
```

Remarks

The minimum length of the PIN.

Pin Maximum Length

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{PinMaxLength} . "\n";
}
```

Remarks

The maximum length of the PIN.

Text Required

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{TextRequired} . "\n";
}
```

Remarks

Indicates if text is required.

Text Minimum Length

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{TextMinLength} . "\n";
}
```

Remarks

The minimum length of the text.

Text Maximum Length

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{TextMaxLength} . "\n";
}
```

Remarks

The maximum length of the text.

From Required

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{FromRequired} . "\n";
}
```

Remarks

Indicates if the from line is required.

From Minimum Length

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{FromMinLength} . "\n";
}
```

Remarks

The minimum length of the from line.

From Maximum Length

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{FromMaxLength} . "\n";
}
```

Remarks

The maximum length of the from line.

Callback Required

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
```

```
    print $row->{CallbackRequired} . "\n";
}
```

Remarks

Indicates if callback is required.

Callback Supported

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{CallbackSupported} . "\n";
}
```

Remarks

Indicates if callback is supported.

Callback Minimum Length

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{CallbackMinLength} . "\n";
}
```

Remarks

The minimum length of the callback.

Callback Maximum Length

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{CallbackMaxLength} . "\n";
}
```

Remarks

The maximum length of the callback.

Type

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{Type} . "\n";
}
```

Remarks

The type.

Smart Message Support

Syntax

```
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{SmartMsg} . "\n";
}
```

Remarks

Smart message support.

Carrier List

The Carrier List is a collection of Carrier objects. An internal Carrier List object is contained in each SMS object. The internal Carrier List object is populated after a successful carrierListSend(). The user can call the carrierList() method to obtain a copy of internal Carrier object, which can be polled for carrier information. See Carrier Object.

Syntax

```
$sms = Net::SMS->new();
@services = $sms->carrierList();
foreach $row (@services)
{
    print $row->{ID} . " " . $row->{Title} . " " . $row->{SubTitle} . "\n";
}
```

Examples

Sending a SMS Message in Perl

```
#!/usr/bin/perl -I ../lib

use Net::SMS;

# Create a new SMS object
my $r = Net::SMS->new();

# Subscriber properties
$r->subscriberID( '123-456-789-12345' );
$r->subscriberPassword( 'Password Goes Here' );

# Message properties
$r->msgPin( "+1 100 510 1234" );
$r->msgFrom( "Demo" );
$r->msgCallback( "+1 100 555 1212" );
$r->msgText( "Hello World from Simplewire!" );

# Send Message
print "Submitting message To Simplewire...\n";
$r->msgSend();

# Check for errors
if ($r->success)
{
    print "Message was successfully sent via Simplewire!\n";
}
else
{
    print "Message was not successfully sent via Simplewire!\n";
    print "Error Code: " . $r->errorCode . "\n";
    print "Error Description: " . $r->errorDesc . "\n";
}
```

Retrieving a Carrier List in Perl

```
#!/usr/bin/perl -I ../lib

use Net::SMS;

# Instantiate new SMS object
my $sms = Net::SMS->new();

$sms->optFields( "selectbox" );

# Send the request now
$sms->carrierListSend();


# Check For Errors
if ($sms->success)
{
    print "Service List successfully downloaded!\n";
}
else
{
    print "Service List could not be successfully downloaded!\n";
    print "Error Code: " . $sms->errorCode . "\n";
    print "Error Description: " . $sms->errorDesc . "\n";
    exit(1);
}

## Grab a service one at a time and plop them into a hash
@services = $sms->carrierList();

foreach $row (@services) {
    print $row->{ID} . " " . $row->{Title} . " " . $row->{SubTitle} . "\n";
}
```

Checking Message Status in Perl

```
#!/usr/bin/perl -I ../lib

use Net::SMS;

# Create a new SMS object
my $r = Net::SMS->new();

# Subscriber properties
$r->subscriberID( '123-456-789-12345' );
$r->subscriberPassword( 'Password Goes Here' );

# Message properties
$r->msgPin( "+1 100 510 1234" );
$r->msgFrom( "Demo" );
$r->msgCallback( "+1 100 555 1212" );
$r->msgText( "Hello World from Simplewire!" );

# Send Message
print "Submitting message To Simplewire...\n";
$r->msgSend();

# Check for errors
if ($r->success)
{
    print "Message was successfully sent via Simplewire!\n";
}
else
{
    print "Message was not successfully sent via Simplewire!\n";
    print "Error Code: " . $r->errorCode . "\n";
    print "Error Description: " . $r->errorDesc . "\n";
}


# And Now Check The Message Status...
print "\nChecking Status of Message...\n";
my $ticketID = $r->msgTicketID();


# Send the request off
print "Submitting Status Request To Simplewire...\n";
$r->msgStatusSend();
```

```
# Check if the status check was successful
if ($r->success)
{
    print 'The status of message #' . $ticketID . " was retrieved.\n";
    print 'Status Code: ' . $r->msgStatusCode . "\n";
    print 'Status Desc: ' . $r->msgStatusDesc . "\n";
}
else
{
    print 'The status of message #' + $ticketID + " could NOT be retrieved.\n";
    print 'Error Code: ' . $r->errorCode . "\n";
    print 'Error Description: ' . $r->errorDesc . "\n";
}
```

Support

Please submit any problems, bug reports, incompatibilities, requests for change, or other comments in our Support section. 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!

Disclaimer

THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. SIMPLEWIRE SHALL NOT BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL OR OTHER DAMAGES FOR USE OF THIS SOFTWARE. SIMPLEWIRE CANNOT GUARANTEE THE DELIVERY OF TEXT-MESSAGES.