redcoal Mobile Internet Developer API (MIDA) for SOAP-XML Web Services

Version 7.11
September 2005

Technical Support: support@redcoal.com
Or visit http://www.redcoal.com/

Table of Contents

1. Introduction

This document provides instructions on how application developers and web designers can access redcoal's mobile services and integrate them into their own solutions. These mobile services exploit redcoal's latest SMS/MMS technologies, and are provided as SOAP compliant XML web services.

The basic services provided include sending text SMS Messages, binary SMS messages such as ring tones, logos, picture messages, and many housekeeping services. Newer services such as sending EMS messages, MMS messages and WAP Push are provided from MIDA version 7.0 onwards.

Access to these services is via redcoal Mobile Internet Developer API (MIDA). The specifications of the APIs are broken down into logical operations, e.g. APIs for sending text SMS messages, binary SMS messages, MMS messages, and miscellaneous services, etc. starting from Chapter 3 of this document. Chapter 2 provides a quick start guide and general procedures for accessing MIDA. In the rest of this chapter, brief descriptions are given to explain the underlying technologies used by redcoal.

1.1 Short Message Service (SMS)

Short Message Service (SMS) is now a familiar service for sending Short Messages (SM). It is originally specified for use in the GSM networks. When sending SM, they are first dispatched to the Short Message Service Centre (SMSC) and later forwarded to the destination. The specifications of SMS involve a number of documents and can be found at http://www.etsi.org/ (this site also provides all the documents on GSM in general).

SMS provides a number of encoding of SM. redcoal supports both 7-bit (the usual encoding of SM) and 8-bit SMS technologies. The 8-bit SM are usually used to encode the so-called binary SM. Binary SM include all the ring tones, logos, picture messages, etc. and also serve as vehicle for sending MMS and WAP Push messages. Most of the common Nokia's proprietary ring tones, logos and picture messages are supported.

In addition, the following 2 kinds of messages are also supported:

- Enhanced Messaging Service (EMS) messages. EMS is a standard developed by Third Generation Partnership Project (3GPP) to embrace and extend the ability to send ring tones, logos and other simple visual messages. For example, most Ericsson ring tones are provided in EMS format. Consequently, these ring tones for most of the modern Ericsson handsets are supported. Details of EMS can be found at http://www.mobileems.com/.

- Motorola's proprietary ring tones. These ring tones are actually plain text messages, but composed in Motorola's proprietary format such that the modern Motorola handsets can recognize them.

EMS messages are sent as binary SM and Motorola ring tones are simply sent as plain text SM by redcoal. Developers wishing to send EMS messages and Motorola's proprietary ring tones, however, must provide these contents already encoded in the correct format. Note that the ability to display these messages depends on the capability of the destined handsets.

redcoal provides *two-way* messaging where the recipient can reply to the first message and the reply message will be delivered to the originator's email address. redcoal hosts a large pool of mobile numbers (MSISDN) that serve as the intermediary stops for sending out original messages and receiving reply messages. A sophisticated path resolution algorithm is used to determine which MSISDN to use as the intermediary stop such that individual reply message will be delivered to their correct destination.

1.2 Multimedia Messaging Service (MMS)

Multimedia Messaging Service (MMS) is the latest technology to send Multimedia Messages (MM). A MM is actually a MIME document with multi-parts, e.g. it can contain plain text, pictures and sound clips, all in one message. When sending MM, same as the SM case, they are first dispatched to and stored at the Multimedia Messaging Service Center (MMSC) and later retrieved by the destination. However, the protocol between them is much more complicated. Details and specification of MMS can be found at http://www.openmobilealliance.org/.

Briefly, when the MMSC receives the MM from the originator and stores it, it sends a notification to the destination first to notify it that there is a MM waiting for retrieval. If the destined handset is set to retrieve the MM (it may be setup in such a way that it ignores such notification), then it retrieves the MM from the MMSC.

The default presentation language for MM is Synchronized Multimedia Integration Language (SMIL). SMIL is a mark-up language, like HTML, for specifying how and when certain multipart contents should be displayed. The original SMIL is a rather large class of language. The latest version is 2.0, which can be downloaded at http://www.w3.org/TR/smil20/. However the early versions of SMIL MMS only support a limited subset of the original SMIL 2.0 language and a limited set of multimedia contents such as static images, voice and sound clips, etc. These are detailed in the MMS conformance specifications at http://www.openmobilealliance.org/ and the 3GPP SMIL document at http://www.3gpp.org/ftp/Specs/archive/26_series/26.234/.

1.3 WAP Push (Service Indication)

redcoal also supports sending WAP Service Indication (SI) messages. These are also called WAP Push messages because they are "pushed" to the destination. SI mainly contains WAP or Web address sent to the destined handset and stored there. Later the user can visit these sites via the WAP browser (and GPRS) of the handset. Details and specification of Wireless Application Protocol (WAP) and WAP Push technologies can be

found at http://www.openmobilealliance.org/.

Note again that the ability to receive and display MM and WAP SI messages depends on the capability of the destined handsets.

1.4 Simple Object Access Protocol (SOAP)

Direct requests to and replies from the various methods provided by MIDA conform to formats specified by the Simple Object Access Protocol (SOAP). SOAP is a lightweight and simple XML-based protocol that is designed to exchange structured and typed information on the Web. The actual specifications of all the methods (their signatures) are contained in the Web Service Definition Language (WSDL) that redcoal provide. An introduction to SOAP and WSDL can be found at http://ww.w3.org/TR/SOAP/ and http://www.w3.org/TR/wsdl/ respectively. Details about XML can be found at http://www.w3.org/XML/.

Many software development environments provide tools for easily accessing SOAP web services, e.g. Microsoft .NET. Examples in VB.NET will be provided in subsequent chapters to explain and illustrate the various methods and general routines.

In previous versions of MIDA, each SOAP method has a corresponding implementation in Microsoft COM based component for easy access to SOAP web services. From MIDA version 7.0 onwards, continual support for COM will cease to be provided and all accesses to mobile services are meant to be via direct SOAP web services. Hence the latest COM based component supported is version 6.0. Although it will still be valid in older developments for backward compatibility, developers are strongly encouraged to access SOAP web services directly. However, application developers are free to write their own latest COM based components for accessing redcoal's newer SOAP web services.

1.5 redcoal Gateways

Behind the scene, the SOAP web services are provided at the redcoal Mobile Internet Gateway (MIG). The gateway dispatches SM via reliable links to a number of Short Message Service Centre (SMSC) using Short Message Peer to Peer Protocol (SMPP). Hence the gateway has the sufficient bandwidth and capacity to process large volume (millions) of SM. For the interesting developers, details and specifications of SMPP can be found at http://www.smpp.org/.

MIG connects to several different manufacturers' SMSC on one side and provides a uniform interface to communicate with applications on the other. Hence it makes it possible and easy for applications to access the various SMSC by using only one interface.

redcoal reaches more than 130 countries and 300 networks all over the world. An up-to-date list of supported countries and networks can be found at http://www.redcoal.com/.

For MMS, developers can provide a number of parts of the MM to the MIG and on the gateway side these parts will be assembled into a single MM. The MM itself will be stored at the redcoal MMS Gateway (MMSG) and a MMS notification message will then be sent to the destination, whereby it will retrieve the actual MM from the MMSG. Hence the MMSG acts as the MMSC.

1.6 Security

At the moment, transmissions between redcoal SOAP web services and user applications over the web and over-the-air (wireless transmissions) are not encrypted, i.e. information is still transmitted as clear text. MIDA only provides basic authentication via user account passwords and serial numbers. However, MIDA will include an alternative version of secure web services using secure HTTP (HTTPS) or SSL, in addition to the basic authentication in the near future. Over-the-air security will not be considered.

2. Getting Started

This chapter provides a quick start guide to accessing redcoal SOAP web services and explains some of the features of MIDA. Example VB.NET codes for accessing web services and other general routines will also be provided to help developers get started more easily. A brief quick start steps are:

1. Obtain a password (*SMSKey*) from redcoal;
2. Generate a serial number;
3. Import the redcoal SOAP web services reference in your development project;
4. Invoke the MIDA methods in your program to access redcoal SOAP web services.

The above steps are explained in more details and illustrated with examples in the following sections.

2.1 SMS Key and Serial Number

All MIDA methods require a serial number and a password (the SMS key) for basic authentication. They are provided as the first two input parameters of *all* methods respectively:

- **strInSerialNo** – the serial number, and
- **strInSMSKey** – the SMS key

To obtain the SMS key, you need to register an account with redcoal. If you have not done so, go to http://www.redcoal.com/ and register your details under the 'Login' section, after which you will be sent via email a SMS key. There are also a number of account settings that affect how the web services behave. For description of these settings, login and check out the 'Services' page.

To obtain the serial number, you need to download the MIDA package. If you have not done so, go to http://www.redcoal.com/ and download it under the 'Developers' section. Run the serial number generation program 'SerialGen.exe' included in the package to generate a serial number for you. The serial number will be stored in the file '*serialno.txt*' in your working directory.

2.2 License

The first time redcoal SOAP web services are accessed, a temporary license will be issued to the user. This license permits the user to access redcoal web services for a free trial period, which is normally 30 days. After this trial period, a one-time license fee will be charged for continuous access of the web services. Alternative agreement between the user and redcoal is possible to waive this license fee. Check out the latest details at http://www.redcoal.com/.

2.3 Error Codes

All method calls return an integer error code (except **RedWebServiceVersion** in Section 8.4). If the call is successful, the error code is 0. Otherwise some positive integer error code will be returned to reflect the failure reason. A complete list of all error codes and their meanings is given in Appendix A.

2.4 Notification

In each of the MIDA '**Send…**' methods explained in the following chapters, a parameter that contains the reply email of the user is mandatory to access that method. This reply email is used for receiving any notification during the processing and dispatch of the requested mobile message. When all the preliminary checks of the user details and parameters of the method call pass, the message will be stored at redcoal gateway, waiting to be dispatched to the SMSC. In the following situations, a notification will be sent to the user reply email:

1. If errors occur during the processing of the stored message at the gateway or dispatch of the message to the SMSC, a notification will be sent with the failure reason.
2. If the message fails to get deposited at the SMSC, it will be stored for later retry. But after a number of continuous failures, redcoal gateway will give up and abort and an abort notification will be sent.
3. If receipts are available from a specific SMSC and a successful receipt is received for the message (which means the message has been delivered to the destination successfully), then a confirmation notification is sent (if such option is enabled in the account setting).
4. If an unsuccessful receipt is received, then redcoal gateway will retry until abort when an abort notification will be sent, or success when a confirmation is sent (if such option is enabled in the account setting).

2.5 Message ID

In each of the MIDA '**Send…**' methods explained in the following chapters, the message successfully deposited at the redcoal server would be given a message ID returned back for future reference. The format of the message ID returned is

```
Destination:MessageID
```

If the same message is sent to more than one destination in the recipient list, then each message sent to each destination in the list has its own unique ID and a list of the message IDs will be returned, with the format

```
Destination_1:MessageID_1,...,Destination_n:MessageID_n
```


in the same order of the recipients specified in the recipient list.

If messages sent to some destinations in the recipient list cannot be deposited for some reason, e.g. the destination is invalid, while other messages sent to other destinations in the list are successfully deposited, then only the successfully deposited message IDs will be returned, with those failed represented as blank. For example, suppose the same message is sent to 3 destinations below:

```
+61041234567,+1234,+61042345678
```

The messages sent to the first and third are successfully deposited, but the one sent to the second fails because the destination is invalid. Then the list of message IDs returned will be:

```
+61041234567:123456,,+61042345678:123457
```

in the same order of the recipients in the recipient list, with the second blank.

2.6 Web Services Access

The specifications of all MIDA methods are contained in WSDL that redcoal provide. These specifications specify the signatures, types and formats of input and output parameters, as well as any return values of the methods, and are needed to correctly access the web services.

The WSDL of the MIDA methods is located at:
http://xml.redcoal.com/soapserver.dll/wsdl/ISoapServer

Some third party SOAP testing web site can be used to test these methods. One of them is the following:

http://www.soapclient.com/soaptest.html

To access redcoal SOAP web services in your development project, you need to provide the WSDL reference, listed above. In the following, a general procedure for referencing redcoal web services is provided for VB.NET project:

1. Open your Visual Studio .NET, create a new 'Visual Basic' project using the 'Windows Application' template, name it and click 'OK'.
2. Now add the web reference to the redcoal web service. A web reference creates the proxy class needed to communicate with the web service. From the Project menu click 'Add Web References'. The Add Web Reference dialog box opens. Enter the address of the WSDL listed above in the address bar, then press 'Enter'. The redcoal web service gets loaded and its information is shown in the dialog box. Moreover the 'Add Reference' button will be enabled.

3. Click the 'Add Reference' button to add this web reference to your project. (If you don't have Visual Studio .NET you will have to use command line program '*wsdl.exe*' to generate the proxy class.)
4. Now in the 'Solution Explorer', expand the 'Web References' folder to note the namespace (com.redcoal.xml) for the redcoal Web reference classes. This name can be changed to any other name, e.g. simply 'Redcoal'.

After adding the web reference classes, a web service instance can be created to access the various methods. Multiple instances can be created and they can be global or local instances. In VB.NET, creating web service instance is as simple as the following (supposing the namespace in step 4 above is renamed as 'Redcoal'):

```
Dim ws As Redcoal.ISOAPServerservice =
New Redcoal.ISOAPServerservice()
```

Then the web service instance '*ws*' can be used to access the various methods as follow, where in the code, '*method*' is any MIDA method name with n parameters and the return value is stored in '*Result*':

```
Dim Result As Integer
'---prepare parameters param_1,...,param_n
Result = ws.method(param_1,...,param_n)
'---check the error code in "Result"
```

2.7 Base64 Byte Stream

Many methods accept file content as one of the parameters, to be included as part of the content of the SM or MM to be delivered to the destination, e.g. the method **SendBinarySMS** expects a parameter that contains the content of, say, a logo or ring tone. These contents are encoded using base64 and transmitted as a byte stream via the web. The following example VB.NET code is provided as a template for converting some file content (the path name of which is '*FilePath*') to a byte array, passed as a parameter of the method **SendBinarySMS**.


```
Imports System.IO
.
.
Dim s1 As FileStream
Dim br As BinaryReader
s1 = New FileStream(FilePath.Text, FileMode.Open,
                    FileAccess.Read)
br = New BinaryReader(s1)
Dim a As Integer = br.BaseStream.Length()
Dim byteRead(a) As Byte
Dim j As Integer
For j = 0 To br.BaseStream.Length() - 1
byteRead(j) = br.ReadByte
Next
br.Close()
.
.
Dim Result As Integer
Result = ws.SendBinarySMS(..., byteRead, ...)
.
.
```

2.8 Format of MIDA Specifications

The following chapters break down all the supported MIDA methods into their logical operations, and describe their functionalities and properties. All methods take as their first 2 parameters the serial number and SMS key respectively, and return an error code (except **RedWebServiceVersion** in Section 8.4), as explained in Section 2.1 and 2.3. The general form of the description of the methods is:

- Synopsis – the form of the method call similar to C++ language
- Description – the use and description of the function of the method
- Parameters – type and description of the input/output parameters. The list of parameters is in table format with columns:
  - Parameter name
  - Parameter type
  - Input and/or output parameter
  - Description
- Related material and special notes (optional)
- Example VB.NET code

3. Sending Text and Binary SMS Messages

3.1 SendTextSMS

Synopsis

```
int SendTextSMS(string strInSerialNo,
                string strInSMSKey,
                string strInRecipients,
                string strInMessageText,
                string strInReplyEmail,
                string strInOriginator,
                int    iInType,
                string &strOutMessageIDs)
```

Description

This is by far the most common method that sends a text SMS message (the content of which is specified in the parameter **strInMessageText**) to the (possibly multiple) mobile destinations as specified in the parameter **strInRecipients**.

The SM can be a simple *one-way* message*,* a *two-way* message, or a *flash* message as specified in the parameter **iInType**.

For one-way message, the text message can only be forwarded to the destination. The recipients cannot reply back to the originator. In two-way SMS messaging, recipients can reply and the reply message is forwarded back to the originator via email as specified in the parameter **strInReplyEmail**, or alternatively retrieved by the originator via the method **GetIncomingMessage** (see Chapter 6). Flash messages are those that are displayed on the screen of the recipients' mobile phone without being stored in the handset's inbox.

The message that arrives at the recipient handset will show the name of the originator specified in the parameter **strInOriginator**. However, the originator may not always be shown at the handset, particularly if the originator contains non-numeric alphabets because some networks/SMSC do not allow alphabets to be displayed as the sender. In such case, the networks/SMSC will substitute the originator with one of their own numbers as the sender, or worse, just reject the message. Moreover, if the message is two-way, the parameter **strInOriginator** will be ignored because the sender will be one of redcoal's numbers for receiving the replies.

Each message deposited for *each* destination in **strInRecipients** will be assigned a unique message identifier, returned via the output parameter **strOutMessageIDs**. Call the method **CheckMessageStatus** (see Section 8.1) to check the status of the message for a particular destination (delivered, processed etc.).

A notification will be sent to the sender via email specified in the parameter **strInReplyEmail** when one of the situations outlined in section 2.4 occurs.

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInRecipients	String	In	Comma separated list of recipients' mobile numbers, preferably in international format, e.g. +4179123123,+6140123123
strInMessageText	String	In	Message content, maximum 160 characters. Messages longer than 160 characters will be cut off.
strInReplyEmail	String	In	Error notifications and message delivery confirmations (if enabled) will be sent to this address, as well as replies to the message if two-way message is sent.
strInOriginator	String	In	The sender displayed at the destination handset. Both numbers and alphanumeric strings are allowed. If the message is two-way, this parameter will be ignored.
iInType	Integer	In	This parameter specifies the type of message to be sent: ❑ 0 = one-way SM ❑ 1 = two-way SM ❑ 2 = flash SM
strOutMessageIDs	String	Out	Comma separated list of message identifiers, each corresponds to the message sent to a destination in the same order specified in **strInRecipients**, e.g. +4179123123:123456, +6140123123:123457

See also

CheckMessageStatus, GetIncomingMessage, SendSMS2

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInRecipients As String =
  "+4179123123,+6140123123"
Dim strInMessageText As String = "A text message"
Dim strInReplyEmail As String = "myemail@mycom.com"
Dim strInOriginator As String = "originator"
Dim iInType As Integer = 0  '---one way message
Dim strOutMessageIDs As String
.
.
Dim Result As Integer
Result = ws.SendTextSMS(strInSerialNo,
                        strInSMSKey,
                        strInRecipients,
                        strInMessageText,
                        strInReplyEmail,
                        strInOriginator,
                        iInType,
                        strOutMessageIDs)
.
.
```


3.2 SendBinarySMS

Synopsis

```
int SendBinarySMS(string       strInSerialNo,
                  string       strInSMSKey,
                  string       strInRecipients,
                  base64Binary strInBinaryContent,
                  string       strInExtraParam,
                  string       strInReplyEmail,
                  int          iInType,
                  string       &strOutMessageIDs)
```

Description

This method sends a *binary* SMS message (the content of which is specified in the parameter **strInBinaryContent**, which is a Base64-encoded byte stream) to the (possibly multiple) mobile destinations as specified in the parameter **strInRecipients**.

A binary SM can be one of the followings, as specified in the parameter **iInType** (a list of the possible formats for these messages and their descriptions is given in Appendix B):

- Operator logo (bitmap)
- Mono ring tone (RTTTL format)
- Polyphonic ring tone (MIDI)
- Picture message (bitmap + text)
- VCard (text)
- WAP Bookmark (text)
- Java game (java archive .jar)

Operator logos, mono ring tones, picture messages, VCards and WAP bookmarks must conform to the format specified in Nokia's *Smart Messaging Protocol*. Consequently, only Nokia's handsets can interpret these binary messages. For handsets of other manufacturers, the alternative method **SendBinarySMSByContent** (see Section 3.3) should be used.

For sending polyphonic ring tones and java games, the actual ring tones or java games are not sent. Instead, they are stored at the redcoal MMS gateway. A special kind of WAP Push message called the *Service Indication (SI)* is used to send a URL of these ring tones or java games to the destinations handsets (see **SendWAPSI** in Section 4.4). Then the user can download these contents from the MMS gateway via WAP. For this reason, the destination handsets must also support WAP. An additional text message that contains the actual URL will also be sent to the destination handsets. User with handset that does not support SI can enter the URL manually into the WAP browser of the handset and retrieve the content.

The purpose of the parameter **strInExtraParam** depends on the content types of the binary message:

- If the message is an operator logo, **strInExtraParam** contains the network with which the destination handset is. The format should be "XXX YY" where XXX is the MCC and YY is the MNC, separated by a white space. A list of supported MCC and MNC is given in Appendix D.
- If the message is a picture message, **strInExtraParam** contains the text of the message while **strInBinaryContent** contains the picture.
- If the message is a WAP bookmark, **strInExtraParam** contains the name of the bookmark in plain text (used as a title for the bookmark) while **strInBinaryContent** contains the actual URL of the bookmark in plain text.
- If the message is a polyphonic ring tone or a java game, **strInExtraParam** contains the name of the content in plain text (used as a title for the content) while **strInBinaryContent** contains the actual content (ring tone or java game).
- For all other types (mono ring tone and VCard), the parameter **strInExtraParam** is ignored and can be empty.

This method can also be called to restore the original operator logo; the current one will effectively be removed. If this is the case, both the parameters **strInBinaryContent** and **strInExtraParam** are ignored and can be empty.

Binary messages are always *one-way* messages. The originator will be one of redcoal's numbers. Hence a parameter **strInOriginator** like that in **SendTextSMS** is absent in this method. But a notification will still be sent to the sender via email specified in the parameter **strInReplyEmail** when one of the situations outlined in section 2.4 occurs.

Each message deposited for *each* destination in **strInRecipients** will be assigned a unique message identifier, returned via the output parameter **strOutMessageIDs**. Call the method **CheckMessageStatus** (see Section 8.1) to check the status of the message for a particular destination (delivered, processed etc.).

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInRecipients	String	In	Comma separated list of recipients' mobile numbers, preferably in international format, e.g. +4179123123,+6140123123
strInBinaryContent	Base64	In	A byte stream encoded with base64 containing the actual binary content. See section 2.6 for how to convert a binary file to a byte stream.
strInExtraParam	String	In	This parameter contains extra information of the binary message, depending on the specific message type as outlined above.
strInReplyEmail	String	In	Error notifications and message delivery confirmations (if enabled) will be sent to this reply email address.
iInType	Integer	In	This parameter specifies the type of message to be sent: ❑ 5506 = operator logo ❑ 5505 = mono ring tone ❑ 5514 = picture message ❑ 9204 = VCard ❑ 9039 = WAP bookmark ❑ 9601 = polyphonic ring tone ❑ 2960 = java game ❑ 255 = remove operator logo
strOutMessageIDs	String	Out	Comma separated list of message identifiers, each corresponds to the message sent to a destination in the same order specified in **strInRecipients**, e.g. +4179123123:123456, +6140123123:123457

See also

CheckMessageStatus, SendBinarySMSByContent, SendWAPSI, SendSMS2

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInRecipients As String =
  "+4179123123,+6140123123"

'---byteRead is a byte array of length (a)
'---that contains the actual content of the
'---operator logo, obtained as in section 2.6
Dim byteRead(a) As Byte

'---the network is Australia Telstra
Dim strInExtraParam As String = "505 01"

Dim strInReplyEmail As String = "myemail@mycom.com"
Dim iInType As Integer = 5506  '---operator logo
Dim strOutMessageIDs As String
.
.
Dim Result As Integer
Result = ws.SendBinarySMS(strInSerialNo,
                          strInSMSKey,
                          strInRecipients,
                          byteRead,
                          strInExtraParam,
                          strInReplyEmail,
                          iInType,
                          strOutMessageIDs)
.
.
```


3.3 SendBinarySMSByContent

Synopsis

```
int SendBinarySMSByContent(string       strInSerialNo,
                           string       strInSMSKey,
                           string       strInRecipients,
                           base64Binary strInBinaryContent,
                           string       strInExtraParam,
                           string       strInReplyEmail,
                           int          iInContentType,
                           int          iInSourceType,
                           string       &strOutMessageIDs)
```

Description

This method has the same functionality of **SendBinarySMS**, which sends a *binary* SMS message (the content of which is specified in the parameter **strInBinaryContent**, which is a Base64-encoded byte stream) to the (possibly multiple) mobile destinations as specified in the parameter **strInRecipients**.

Unlike **SendBinarySMS**, which mostly accepts contents conformed to the formats specified by Nokia's *Smart Messaging Protocol*, this method accepts additional content types for handsets by other manufacturers. Hence this method can be considered as an extended version of and is backward compatible with **SendBinarySMS**.

The parameter **iInContentType** specifies the actual type of message to be delivered. It is the same as the parameter **iInType** of **SendBinarySMS**. The parameter **iInSourceType** specifies the source type of the supplied content in the parameter **strInBinaryContent** so appropriate processing is used to process the content. Accepted source types are (a list of these types and their descriptions as well as their limitations is given in Appendix C):

- Hex coded byte stream
- Plain text
- Bitmap (.bmp)
- standard over-the-air picture format (.ota)
- mono ring tones in RTTTL format (.rtttl)
- MIDI (.mid)
- Java archive (.jar)
- NA (no specific type, treated depending on **iInContentType**)

If the source type is plain text, then **strInBinaryContent** should be just plain text, although the text will be sent as Base64-encoded byte stream. Thus the method outlined in section 2.6 that converts content to a byte array should be applied to the text.

Note that certain message content types can only accept certain source types. For instance, operator logos cannot accept mono ring tones in RTTTL format. If these two types are not compatible, an error code of 1 will be returned.

This method also compensates the limitations of **SendBinarySMS**, by allowing other protocols' binary messages to be sent, e.g. *EMS* (must be pre-encoded into a HEX string) and Motorola ring tones (plain text with Motorola's proprietary ring tone format). For these messages, the parameter **strInExtraParam** is ignored and can be empty.

For the descriptions of other parameters and features, refer to **SendBinarySMS** in Section 3.2.

Parameters

Parameters	Type	In/Out	Description
All the parameters except the ones below are the same as those in **SendBinarySMS**. Refer to Section 3.2 for the descriptions of those parameters.			
iInContentType	Integer	In	This parameter is the same as **iInType** of **SendBinarySMS** with an additional type for EMS: ❑ 256 = EMS ❑ 848 = Motorola ring tone
iInSourceType	Integer	In	This parameter specifies the source type of the content: ❑ 0 = NA ❑ 1 = hex coded byte stream ❑ 2 = plain text ❑ 11 = bitmap ❑ 12 = OTA picture ❑ 21 = RTTTL ring tone ❑ 22 = MIDI polyphonic ❑ 23 = java archive

See also

CheckMessageStatus, SendBinarySMS, SendSMS2

Example

See the example of **SendBinarySMS** in Section 3.2.

3.4 SendSMS2

Synopsis

```
int SendSMS2(string       strInSerialNo,
             string       strInSMSKey,
             string       strInRecipients,
             base64Binary strInContent1,
             string       strInContent2,
             string       strInOriginator,
             string       strInReplyEmail,
             string       strInScheduledUTCDateTime,
             int          iInMessageType,
             int          iInOptions,
             string       &strOutMessageIDs)
```

Description

This method sends either a text or binary SMS message to the (possibly multiple) mobile destinations as specified in the parameter **strInRecipients**. It basically combines the functions of **SendTextSMS**, **SendBinarySMS**, and scheduled messaging (see Chapter 5) into one function. The parameters **strInRecipients**, **strInOriginator** and **strInReplyEmail** are as in these methods and their descriptions can be found under their corresponding sections.

For text SMS messages, the parameter **strInContent1** contains the message text and the parameter **strInContent2** is ignored. The parameter **strInContent1** has the same functionality as the parameter **strInMessageText** of **SendTextSMS** and thus should be plain text although it is sent as Base64-encoded byte stream. The method outlined in section 2.6 that converts content to a byte array should be applied to the text. The type of text SMS messages is specified in **iInMessageType**. For the description of the types of text SMS messages and of sending them, please consult **SendTextSMS**.

For binary SMS messages, the parameter **strInContent1** contains the binary content and **strInContent2** contains the extra information, if any, just like the parameters **strInBinaryContent** and **strInExtraParam** of the method **SendBinarySMSByContent** respectively. The parameter **strInOriginator** is ignored and the binary message type and source type are specified in **iInMessageType** and **iInOptions** respectively, just like the parameters **iInContentType** and **iInSourceType** of **SendBinarySMSByContent** respectively. If no source type is specified, then the binary content is treated as default type, as in **SendBinarySMS**. Unlike **SendBinarySMSByContent,** which complains about incompatible content type and source type, if incompatible source type is specified in **iInOptions** for the message type specified in **iInMessageType**, then the source type is ignored, and the binary content will be treated as the default type for the specified message type. For the description of the different types of binary SMS messages and of

sending them, please consult **SendBinarySMS** and **SendBinarySMSByContent**.

The parameter **iInOptions** also includes the following additional options:

- Confirmation from SMSC (0x100 = 256): if specified, a confirmation will be sent to **strInReplyEmail** when the message is successfully dispatched to the SMSC.
- Confirmation from handset (0x200 = 512): if specified, a confirmation will be sent to **strInReplyEmail** when the message is successfully delivered to the destination handsets. This is the case if only the SMSC provides feedback status of the message.
- Testing mode (0x1000 = 4096): if specified, the message is only treated as a testing message and will not be dispatched to the SMSC for delivery to the destination handsets. However, an error code and the message ID will still be returned and the status of the message can be checked via **CheckMessageStatus** (see Section 8.1), which will return "TESTING". This option is suitable for testing if the request can be processed successfully without actually sending the message.

The above options will override the default account settings and hence can be specified on a message-by-message basis without changing the account settings. The various options can be combined in **iInOptions** just by adding up the various options. For example, if both confirmations from SMSC and handset are requested for the binary message that contains the OTA picture (source type 12), then 256+512+12 = 780 should be specified in **iInOptions**.

A new parameter, **strInScheduledUTCDateTime**, is added for scheduling the message for later delivery, similar to the method **EnterScheduleExt** (see Section 5.2). So with this parameter, binary SMS messages can also be scheduled for future delivery. This parameter contains a string that specifies the date time in UTC (GMT) when the message should be delivered. The format of the string is

YYYYMMDD:HH:MM:SS

- YYYY is the year
- MM is the month (01-12) with leading 0,
- DD is the day of month (01-31) with leading 0
- HH:MM:SS are the hour of day, minutes of hour and seconds of minute in 24 hours format, all with leading 0
- The colon ":" must also be included as specified

For example "20041224:21:30:00" specifies 24 Dec 2004 9:30pm in GMT. Users must take care of time zone difference and daylight saving when specifying **strInScheduledUTCDateTime**, otherwise inaccurate time of delivery may result. Some source for conversion between GMT and local time can be obtained at http://www.twinsun.com/tz/tz-link.htm. An online converter can also be accessed at http://www.worldtimeserver.com/. If **strInScheduledUTCDateTime** is empty or the date time specified in it has passed, then the message will be delivered as soon as it has been

processed successfully.

Each message deposited for *each* destination in **strInRecipients** will be assigned a unique message identifier, returned via the output parameter **strOutMessageIDs**. Call the method **CheckMessageStatus** (see Section 8.1) to check the status of the message for a particular destination (delivered, processed etc.).

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInRecipients	String	In	Comma separated list of recipients' mobile numbers, preferably in international format, e.g. +4179123123,+6140123123
strInContent1	Base64	In	A byte stream encoded with base64 containing the message text for text messages and binary content for binary messages. See section 2.6 for how to convert to a byte stream.
strInContent2	String	In	This parameter is just the parameter **strInExtraParam** of **SendBinarySMSByContent**.
strInOriginator	String	In	This parameter is the same parameter **strInOriginator** of **SendTextSMS**.
strInReplyEmail	String	In	Error notifications and message delivery confirmations (if specified in **iInOptions**) will be sent to this reply email address.
strInScheduledUTC DateTime	String	In	This parameter specifies the UTC date time in the format specified above for future delivery. Message will be sent immediately if this parameter is empty.
iInMessageType	Integer	In	This parameter specifies the type of the text or binary SMS message to be sent.

iInOptions	Integer	In	This parameter specifies the source type of binary content (see **iInSourceType** of **SendBinarySMSByContent**) and the options: ❑ SMSC confirmation (256) ❑ Handset confirmation (512) ❑ Testing mode (4096)
strOutMessageIDs	String	Out	Comma separated list of message identifiers, each corresponds to the message sent to a destination in the same order specified in **strInRecipients**, e.g. +4179123123:123456, +6140123123:123457

See also

CheckMessageStatus, SendTextSMS, SendBinarySMS, SendBinarySMSByContent, EnterScheduleExt

Example

See the examples in **SendTextSMS**, **SendBinarySMS** and **EnterScheduleExt**.

4. Sending MMS and WAP Push Messages

4.1 CreateMMS

Synopsis

```
int CreateMMS(string strInSerialNo,
              string strInSMSKey,
              string &strInOutTransactionID)
```

Description

This method allocates resources for creating an empty MMS message to be sent later using the method **SendMMS** (see Section 4.3) after the individual parts are added to it using the method **AddMMSContent** (see Section 4.2).

Because MMS messages can contain multiple parts (they are MIME multipart documents), the sending of a MMS message is broken down into 3 steps:

1. Create an empty MMS
2. Add individual parts to the MMS
3. Send the MMS

This method allocates the necessary resources and creates an empty MMS message with a unique transaction ID. A user-defined unique ID can be set in the parameter **strInOutTransactionID** to uniquely identify this MMS message. Alternatively, this parameter should be set to blank and the method will automatically allocate a unique ID for this message, returned again via the parameter **strInOutTransactionID**.

redcoal supports two types of MMS messages. The first type of MMS message has a MIME content type '*multipart/mixed*' where no specific presentation is required. The various contents of the MMS message (identified by the transaction ID), e.g. plain text, pictures and sound/speech clips, are then added one at a time, by calling the method **AddMMSContent**. When all contents are added, the method **SendMMS** is used to send the whole MMS message (in fact, only a notification message is sent to the destination handset, when the handset can retrieve the actual MMS message from the MMS gateway). The contents of the message will be displayed on the destination handset in the same order. Hence the order of addition determines the order of content presentation.

The second type of MMS message has a MIME content type '*multipart/related*' where specific presentation is specified in a user-provided SMIL document, which is also part of the MMS message. Hence the SMIL document is mandatory for sending SMIL MMS message and must be added to the MMS message using the method **AddMMSContent** as well. Other parts are added using **AddMMSContent** in the same way as for the first

type of MMS message. However the order of addition is not important here since the order of presentation is specified in the SMIL document.

The type of MMS message created is not decided in this method since both types are created the same way. The type of MMS message to be sent is actually decided when calling the method **SendMMS**, when it finds a SMIL document previously added using **AddMMSContent**, the MMS message will be sent as the '*multipart/related*' SMIL MMS message. Otherwise it is sent as the '*multipart/mixed*' MMS message.

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInOutTransactionID	String	In/Out	The ID that uniquely identifies the MMS message.

See also

AddMMSContent, SendMMS

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"

'---an empty transaction ID is used
Dim strInOutTransactionID As String
.
.
Dim Result As Integer
Result = ws.CreateMMS(strInSerialNo,
                      strInSMSKey,
                      strInOutTransactionID)
'---an automatically generated transaction ID
'---is given out in strInOutTransactionID
.
.
```


4.2 AddMMSContent

Synopsis

```
int AddMMSContent(string       strInSerialNo,
                  string       strInSMSKey,
                  string       strInTransactionID,
                  base64Binary strInContent,
                  string       strInContentType
                  string       strInContentLocationOrID)
```

Description

This method adds one content part to the MMS message (previously created by the method **CreateMMS**) identified by the transaction ID in the parameter **strInTransactionID**.

The transaction ID **strInTransactionID** is either the user supplied ID used in the previous call to the method **CreateMMS**, or the returned ID from the previous call to **CreateMMS**.

There is practically no limit as to how many parts can be added to a single MMS message, but this is subject to the limit imposed by the wireless operator of the destination handset. An earlier common denominator of the MMS message size was 30K, but this has increased, and is still increasing, to some larger size, depending on the wireless operators.

For '*multipart/mixed*' MMS message, the display order of the contents on the screen of the destination handset is the same as the order of addition. For '*multipart/related*' SMIL MMS message, the presentation is determined by the user-supplied SMIL document, also added using this method, but the order of addition of various content parts is irrelevant.

Each part has its own content, specified in the parameter **strInContent**, which is a Base64-encoded byte stream. The content type of the content is specified in the parameter **strInContentType**. Currently supported types are:

- text/plain – plain text
- image/gif – GIF pictures (.gif)
- image/jpeg – JPEG pictures (.jpg, .jpe, .jpeg)
- image/bmp – Bitmap pictures (.bmp), subject to handset interpretation
- audio/midi – MIDI sound clips (.mid, .midi)
- audio/amr – adaptive multi-rate (AMR) voice clips, see IETF RFC 3267 at http://www.ietf.org/
- application/smil – SMIL document conforming to MMS standard

If the content type is plain text or SMIL document, then **strInContent** should be just plain

text, although the text will still be encoded as base64 byte stream.

For SMIL MMS message, there should be at least one SMIL document present by which the internal system recognizes the type of MMS message to be sent. Subsequent SMIL documents can also be added but only the first SMIL document will be used as the starting document of the whole MMS message, i.e. the first part of the '*multipart/related*' MMS message.

There are basically two ways to refer to different content parts of the MMS message within the SMIL document: by *Content-Location* or *Content-ID*:

- For *Content-Location*, the URL specifies the location of the content within the MMS message, usually the filename of the content.
- For *Content-ID*, the URL specifies the ID of the content within the MMS message, in the format "`cid:`content-id" where "`cid:`" is the scheme (see IETF RFC 2392) and "content-id" is some user-defined ID for the specified content.

These URL must be specified within the MMS message as well and must match those specified within the SMIL document. Hence a URL must be specified for each content part (except the SMIL document itself) within the SMIL MMS message, in the parameter **strInContentLocationOrID**. These URL will be encoded appropriately within the MMS message. The type of URL will be recognized as follows:

- If the parameter **strInContentLocationOrID** contains the format "`cid:`content-id", then it is treated as a *Content-ID*.
- Otherwise, if the parameter **strInContentLocationOrID** contains some text without the scheme "`cid:`" then it is treated as a *Content-Location*.

For '*multipart/mixed*' MMS message, the *Content-Location* or *Content-ID* specified in **strInContentLocationOrID** is optional because the various content parts are displayed in strict order without need for any reference. However, if the *Content-Location* or *Content-ID* is specified, it will be encoded within the MMS message. This will not alter the presentation but can be used to name the different content parts within the MMS message. Some handsets will use these default names when asked to save the individual content parts inside the handset stores.

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInTransactionID	String	In	The ID that uniquely identifies the MMS message.
strInContent	Base64	In	A byte stream encoded with base64 containing the actual content. See section 2.6 for how to convert a binary file to a byte stream.
strInContentType	String	In	The content type of the content in **strInContent**. The type can be specified as one of the following: ❑ text/plain ❑ image/gif ❑ image/jpeg ❑ image/bmp ❑ audio/midi ❑ audio/amr ❑ application/smil
strInContentLocationOrID	String	In	The *Content-Location* or the *Content-ID* in the format "`cid:`content-id" of the content part.

See also

CreateMMS, SendMMS

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInOutTransactionID As String
.
.
'---CreateMMS must be called first to get the
'---returned transaction ID in strInOutTransactionID
.
.
'---a JPEG picture will be added
'---byteRead is a byte array of length (a)
'---that contains the content of the JPEG file,
'---obtained as in section 2.6
Dim byteRead(a) As Byte
Dim strInContentType As String = "image/jpeg"
'---use the filename as the Content-Location
Dim strInContentLocationOrID As String = "pic.jpg"

Dim Result As Integer
Result = ws.AddMMSContent(strInSerialNo,
                          strInSMSKey,
                          strInOutTransactionID,
                          byteRead,
                          strInContentType,
                          strInContentLocationOrID)
.
.
```


4.3 AddBase64MMSContent

Synopsis

```
int AddBase64MMSContent(string  strInSerialNo,
                        string  strInSMSKey,
                        string  strInTransactionID,
                        string  strInBase64Content,
                        string  strInContentType
                        string  strInContentLocationOrID)
```

Description

This method has exactly the same function as **AddMMSContent**. This method provides an alternative way to add content parts to MMS message. The only difference between **AddMMSContent** and **AddBase64MMSContent** is:

- In **AddMMSContent**, the input parameter **strInContent** is a Base64-encoded *byte stream*.
- In **AddBase64MMSContent**, the input parameter **strInBase64Content** is a Base64-encoded *string*.

The content needs to be pre-Base64-encoded by the user and the result (a string) is then put in the parameter **strInBase64Content**. The encoding is needed even for plain text content.

If the parameter **strInBase64Content** is not Base64-encoded, then a return code of 14 will be returned by this method.

Parameters

Parameters	Type	In/Out	Description
All the parameters except **strInBase64Content** are the same as those of **AddMMSContent**. Refer to Section 4.2 for the descriptions of those parameters.			
strInBase64Content	String	In	A string encoded with base64 containing the actual content.

See also

AddMMSContent

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInOutTransactionID As String
.
.
'---CreateMMS must be called first to get the
'---returned transaction ID in strInOutTransactionID
.
.
'---a JPEG picture will be added
Dim strInBase64Content As String
'---strInBase64Content should contain
'---the base64-encoded string of the JPEG file
Dim strInContentType As String = "image/jpeg"
'---use the filename as the Content-Location
Dim strInContentLocationOrID As String = "pic.jpg"

Dim Result As Integer
Result = ws.AddBase64MMSContent(
              strInSerialNo,
              strInSMSKey,
              strInOutTransactionID,
              strInBase64Content,
              strInContentType,
              strInContentLocationOrID)
.
.
```


4.4 SendMMS

Synopsis

```
int SendMMS(string strInSerialNo,
            string strInSMSKey,
            string strInTransactionID,
            string strInRecipients,
            string strInSubject,
            string strInReplyEmail,
            string &strOutMessageIDs)
```

Description

This method sends a *MMS-notification* message for the MMS message identified by the transaction ID in the parameter **strInTransactionID**, to the (possibly multiple) mobile destinations as specified in the parameter **strInRecipients**.

The MMS-notification message is just a notification to the destination handsets, upon receiving the handset will retrieve the actual MMS message (if the handset is setup to retrieve MMS message) from the redcoal MMS Gateway.

The MMS message identified by **strInTransactionID** must be previously created using the method **CreateMMS** and have at least one content part added to it using the method **AddMMSContent**. If no content part is previously added to the MMS message, this method will not send any MMS message and will return error code 44 (empty MMS message content).

The type of MMS message depends on the contents previously added. If a SMIL document is added, then the MMS message is of type '*multipart/related*'. If no SMIL document is added before, then the MMS message is of type '*multipart/mixed*'.

An optional parameter **strInSubject** is used to give the MMS message a subject. The use of this parameter is just like the subject field of emails. If **strInSubject** is blank, then a default subject "MMS" will be substituted.

The email address of the sender specified in the parameter **strInReplyEmail** will be used as the sender of the MMS message, if the "include email address" setting in the account setting under the 'Services' page of the login section is enabled. If this setting is disabled, the email address in **strInReplyEmail** will not be included and so the sender of the MMS message will be anonymous. A notification will be sent to the sender via email specified in the parameter **strInReplyEmail** when one of the situations outlined in section 2.4 occurs.

Each MMS-notification (not the actual MMS message which is identified by the transaction ID) deposited for *each* destination in **strInRecipients** will be assigned a

unique message identifier, returned via the output parameter **strOutMessageIDs**. Call the method **CheckMessageStatus** (see Section 8.1) to check the status of the message for a particular destination (downloaded, processed etc.).

Some wireless operators have a restricting policy that prohibit their subscribers from retrieving MMS messages from 3rd party MMS server such as redcoal's MMS gateway. The consequence is that the destination handset will not be able to retrieve the actual MMS message.

To get around this problem, redcoal has devised to send a WAP Push version of the MMS message to the destination handset if redcoal's MMS gateway has not detected that the destination handset has retrieved the MMS message stored on it within 3 minutes after the MMS notification message has been sent. If the actual MMS message has been retrieved within 3 minutes, then this extra WAP Push version will not be sent.

The WAP Push version is sent as a Service Indication (see **SendWAPSI** in Section 4.4). The user of the destination handset can then retrieve a WML-encoded version of the MMS message via the URL specified in the WAP Push message. This is all done within the same function of **SendMMS** so there is nothing more to be done at the user's side. Note this feature is only available for MMS messages of type '*multipart/mixed*'.

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInTransactionID	String	In	The ID that uniquely identifies the MMS message.
strInRecipients	String	In	Comma separated list of recipients' mobile numbers, preferably in international format, e.g. +4179123123,+6140123123
strInSubject	String	In	The subject of the MMS message.
strInReplyEmail	String	In	Used as the sender of the MMS message (if enabled). Error notifications and message delivery confirmations (if enabled) will be sent to this reply email address.

strOutMessageIDs	String	Out	Comma separated list of message identifiers, each corresponds to the message sent to a destination in the same order specified in **strInRecipients**, e.g. +4179123123:123456, +6140123123:123457

See also

CheckMessageStatus, CreateMMS, AddMMSContent, SendWAPSI

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInOutTransactionID As String
.
.
'---CreateMMS must be called first to get the
'---returned transaction ID in strInOutTransactionID
'---then various content parts should also be
'---added to the MMS message by AddMMSContent
.
.
Dim strInRecipients As String =
  "+4179123123,+6140123123"
Dim strInSubject As String = "Some Subject"
Dim strInReplyEmail As String = "myemail@mycom.com"
Dim strOutMessageIDs As String
Dim Result As Integer
Result = ws.SendMMS(strInSerialNo,
                    strInSMSKey,
                    strInOutTransactionID,
                    strInRecipients,
                    strInSubject,
                    strInReplyEmail,
                    strOutMessageIDs)
.
.
```

4.5 SendWAPSI

Synopsis

```
int SendWAPSI(string strInSerialNo,
              string strInSMSKey,
              string strInRecipients,
              string strInURL,
              string strInContent,
              string strInReplyEmail,
              string &strOutMessageIDs)
```

Description

This method sends a special kind of WAP Push message called *Service Indication (SI)* to the (possibly multiple) mobile destinations as specified in the parameter **strInRecipients**.

The destination handsets must support WAP 1.2 or later version. The SI message is like a WAP bookmark message where it specifies the URL of a resource (in the parameter **strInURL**) that can be downloaded via WAP. This is useful, e.g. to allow user to download pictures or ring tones without manually typing in the URL in the WAP browser of the handset. The parameter **strInContent** contains the name (title) of the URL.

A notification will be sent to the sender via email specified in the parameter **strInReplyEmail** when one of the situations outlined in section 2.4 occurs.

Each SI message deposited for *each* destination in **strInRecipients** will be assigned a unique message identifier, returned via the output parameter **strOutMessageIDs**. Call the method **CheckMessageStatus** (see Section 8.1) to check the status of the message for a particular destination (delivered, processed etc.).

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInRecipients	String	In	Comma separated list of recipients' mobile numbers, preferably in international format, e.g. +4179123123,+6140123123
strInURL	String	In	The URL (WAP bookmark) of the resource in plain text.
StrInContent	String	In	The name (title) of the URL.

strInReplyEmail	String	In	Error notifications and message delivery confirmations (if enabled) will be sent to this reply email address.
strOutMessageIDs	String	Out	Comma separated list of message identifiers, each corresponds to the message sent to a destination in the same order specified in **strInRecipients**, e.g. +4179123123:123456, +6140123123:123457

See also

CheckMessageStatus

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInRecipients As String =
  "+4179123123,+6140123123"
Dim strInURL As String = "http://wap.mycom"
Dim strInContent As String = "A WAP SI"
Dim strInReplyEmail As String = "myemail@mycom.com"
Dim strOutMessageIDs As String
.
.
Dim Result As Integer
Result = ws.SendWAPSI(strInSerialNo,
                      strInSMSKey,
                      strInRecipients,
                      strInURL,
                      strInContent,
                      strInReplyEmail,
                      strOutMessageIDs)
.
.
```


5. Scheduled Messaging

5.1 EnterSchedule

Synopsis

```
int EnterSchedule(string strInSerialNo,
                  string strInSMSKey,
                  string strInRecipients,
                  string strInMessageText,
                  string strInReplyEmail,
                  string strInOriginator,
                  double dInDateTime,
                  double dInRefTime,
                  int    iInType)
```

Description

This method schedules a text SMS message (the content of which is specified in the parameter **strInMessageText**) to be sent in the future to the (possibly multiple) mobile destinations as specified in the parameter **strInRecipients**. For scheduling a binary SMS message, user should use the method **SendSMS2** in Section 3.4. Currently, there is no provisioning of scheduling MMS messages.

The SM can be a simple *one-way* message, a *two-way* message, or a *flash* message as specified in the parameter **iInType**. For the descriptions of these message types, as well as of the parameters **strInReplyEmail** and **strInOriginator**, see **SendTextSMS** in Section 3.1.

The future date time when the scheduled message should be sent is specified in the parameter **dInDateTime**, which must be a number of type double. There are conversion functions that can convert a date time object or a date time string to double number in many development environment. Consult the appropriate programming reference manual for details.

The other parameter **dInRefTime** is used as a reference time, also must be a number of type double. It is basically the user local time and can be obtained by first getting the local time and then converting it to double number. This reference time is used to correct any time difference around the globe. For instance, a message is scheduled to be sent 30 hours from now in UK. The time difference between UK and the redcoal gateway at Sydney is +10 hour (+11 if daylight saving is on). If only **dInDateTime** is used, the message will be sent 10 hours before the actual scheduled time. But with **dInRefTime**, the message will be sent at time (**dInDateTime** – **dInRefTime**) to correct for the time difference.

Parameters

Parameters	Type	In/Out	Description
All the parameters except the ones below are the same as those in **SendTextSMS**. Refer to that section for the descriptions of those parameters.			
dInDateTime	Double	In	This is the date time when the message is scheduled to be sent, in double number format.
dInRefTime	Double	In	This is the user local date time used as a reference time to correct time difference, in double number format.

See also

SendTextSMS, SendSMS2, EnterScheduleExt

Notes

This method is superseded by the method **EnterScheduleExt**. It is only provided here for backward compatibility.

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInRecipients As String =
  "+4179123123,+6140123123"
Dim strInMessageText As String = "A text message"
Dim strInReplyEmail As String = "myemail@mycom.com"
Dim strInOriginator As String = "originator"
Dim iInType As Integer = 0  '---one way message

'---dInDateTime contains the future date time
'---dInRefTime contains the local date time
Dim dInDateTime As Double
Dim dInRefTime As Double
.
.
Dim Result As Integer
Result = ws.EnterSchedule(strInSerialNo,
                          strInSMSKey,
                          strInRecipients,
                          strInMessageText,
                          strInReplyEmail,
                          strInOriginator,
                          dInDateTime,
                          dInRefTime,
                          iInType)
.
.
```


5.2 EnterScheduleExt

Synopsis

```
int EnterScheduleExt(string strInSerialNo,
                     string strInSMSKey,
                     string strInRecipients,
                     string strInMessageText,
                     string strInReplyEmail,
                     string strInOriginator,
                     string dInDateTime,
                     string dInRefTime,
                     string iInType,
                     string &strOutMessageIDs)
```

Description

This method schedules a text SMS message (the content of which is specified in the parameter **strInMessageText**) to be sent in the future to the (possibly multiple) mobile destinations as specified in the parameter **strInRecipients**.

This method has the same functionality as **EnterSchedule**, with an additional output parameter **strOutMessageIDs** that returns the scheduled message identifiers. The message identifiers returned then permit future status checking by **CheckMessageStatus** (see Section 8.1) or schedule deletion by **DeleteSchedule** (see Section 5.3). For the descriptions of all other parameters, see the section for the method **EnterSchedule**.

Each scheduled message deposited for *each* destination in **strInRecipients** will be assigned a unique message identifier, returned via the output parameter **strOutMessageIDs**. Call the method **CheckMessageStatus** to check the status of the message for a particular destination (delivered, processed etc.).

Parameters

Parameters	Type	In/Out	Description
All the parameters except the one below are the same as those in **EnterSchedule**. Refer to that section for the descriptions of those parameters.			
strOutMessageIDs	String	Out	Comma separated list of message identifiers, each corresponds to the message sent to a destination in the same order specified in **strInRecipients**, e.g. +4179123123:123456, +6140123123:123457

See also

CheckMessageStatus, DeleteSchedule, EnterSchedule, SendTextSMS, SendSMS2

Example

See the example in the section of **EnterSchedule**.

5.3 DeleteSchedule

Synopsis

```
int DeleteSchedule(string strInSerialNo,
                   string strInSMSKey,
                   string strInMessageIDs)
```

Description

This method deletes the schedule of a message previously entered by **EnterScheduleExt** or **SendSMS2**, identified by the message identifier **strInMessageIDs**.

The parameter **strInMessageIDs** is the output parameter **strOutMessageIDs** of the method **EnterScheduleExt** or **SendSMS2**, that contains for each comma separated destination handset, a unique message ID. The format of this output parameter should be passed on, as is, to the input parameter **strInMessageIDs** of this method.

The scheduled messages are deleted by this method only if they have not been dispatched to the SMSC successfully, i.e. only if the scheduled messages are still in the status of “PENDING”, “RETRY”, “FAIL” and “FUTURE”.

Note that only the *schedule* associated with the messages identified by **strInMessageIDs** is deleted. The actual messages are still stored at the system, but with the status, **CANCELLED**, such that the user can still check the status of these messages by **CheckMessageStatus** (see Section 8.1).

Parameters

Parameters	Type	In/Out	Description
StrInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInMessageIDs	String	In	Comma separated list of message identifiers obtained from **EnterScheduleExt**, e.g. +4179123123:123456, +6140123123:123457

See also

CheckMessageStatus, EnterScheduleExt, SendSMS2

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInOutMessageIDs As String
.
.
'---
'---EnterScheduleExt or SendSMS2 are called
'---to enter the scheduled messages
'---and the message identifiers
'---are stored in strInOutMessageIDs
.
.
Dim Result As Integer
Result = ws.DeleteSchedule(strInSerialNo,
                           strInSMSKey,
                           strInOutMessageIDs)
.
.
```


6. Receiving Incoming Messages

6.1 GetIncomingMessage

Synopsis

```
int GetIncomingMessage(string strInSerialNo,
                       string strInSMSKey,
                       string strInReplyEmail,
                       string &strOutSender,
                       string &strOutMessageContent,
                       string &strOutTimeStamp,
                       int    &iOutMessagesLeft)
```

Description

This method retrieves SMS replies (of previously sent *two-way* messages) sent to the email address specified in the parameter **strInReplyEmail**.

By default, the SMS replies are already forwarded to the email address of the sender in **strInReplyEmail**. This method provides an additional programmatic way to retrieve SMS replies such that user applications can display these replies in their own manner.

All SMS replies to the specified email address in **strInReplyEmail** that are not yet retrieved via this method will be retrieved. However, only the first one will be returned. Subsequent calls to this method retrieve the rest. The output parameter **iOutMessagesLeft** returns how many more replies (excluding the current one) need to be retrieved so the user knows how many subsequent calls to this method should be made.

The following information of the reply will be returned:

- Sender of the reply (receiver of the previous *two-way* message) in the output parameter **strOutSender**.
- Actual content of the reply message in the output parameter **strOutMessageContent**.
- The date time of the reply received by the gateway in the output parameter **strOutTimeStamp**.

If a call is made to this method when there is no more reply, all the above 3 output parameters are left unchanged and **iOutMessagesLeft** will be 0.

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInReplyEmail	String	In	The reply email address to which the retrieved SMS replies are sent.
strOutSender	String	Out	The sender of the reply (the receiver of the previous two-way message).
strOutMessageContent	String	Out	The content of the reply.
strOutTimeStamp	String	Out	The date time of the reply.
iOutMessagesLeft	Integer	Out	The number of replies to be retrieved (excluding this one).

See also

SendTextSMS

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInReplyEmail As String = "myemail@mycom.com"
Dim strOutSender As String
Dim strOutMessageContent As String
Dim strOutTimeStamp As String
Dim iOutMessagesLeft As Integer = 0
.
.
Dim Result As Integer
DO
  Result = ws.GetIncomingMessage(
                strInSerialNo,
                strInSMSKey,
                strInReplyEmail,
                strOutSender,
                strOutMessageContent,
                strOutTimeStamp,
                iOutMessagesLeft)

  IF Result = 0 THEN
    '---read the reply message
  ELSE
    '---error handling
    EXIT DO
  END IF
LOOP WHILE iOutMessagesLeft > 0
.
.
```


7. Groups Management

7.1 CreateGroup

Synopsis

```
int CreateGroup(string strInSerialNo,
                string strInSMSKey,
                string strInGroupName,
                string strInGroupMembers,
                int    &iOutMembersCreated)
```

Description

This method creates a group for the user with the name specified in **strInGroupName**, that can be later used for multicasting messages.

Users can multicast messages to a group of destination handsets by specifying the group name in the parameter **strInRecipients** of the various '**Send…**' methods or scheduling methods. This is a very convenient way of specifying a large number of destinations.

Before users can specify a group name to send, a group needs to be created first. A group can either be created manually by redcoal upon request or by calling this method. The parameter **strInGroupName** specifies the name of the group to be created. Any existing group of the user with the same name as **strInGroupName** will be overwritten.

The members of the group are specified in the parameter **strInGroupMembers**. This parameter is a string in the following format:

```
<Member Name 1>,<Mobile Number 1>[,Attributes];...;
<Member Name N>,<Mobile Number N>[,Attributes]
```

Members are separated by semi-colon "`;`". Each member has a number of fields, separated by comma "`,`":

- The first field is always the name of the member, which can be empty but the trailing comma is still required.
- The second field is always the valid destination mobile number of the member, which can also be empty (although it is pointless to have empty mobile number). If the mobile number is invalid, then this field will also be left empty.
- The rest of the fields "`Attributes`" is optional. The format of it is

  ```
  Attribute_1=Value_1,...,Attribute_N=Value_N
  ```


where each of the "`Attribute_i`" is the name of an attribute for this member and "`Value_i`" is the corresponding value of this attribute for this member. So it is flexible for the user to choose whatever number of attributes and their values to include for each of the members. These attribute values are useful for *mail merge* when the user multicasts a message to this group with all attribute names in the text of the message replaced by their corresponding values for each member of the group. Users not concerned of this can just ignore all these attributes.

This method returns the number of members actually created for this group in the output parameter **iOutMembersCreated**. If there is none created, i.e. **iOutMembersCreated** contains 0, then the group itself is not created. If the group name specified in **strInGroupName** is empty, an error code of 31 will be returned. If the member list specified in **strInGroupMembers** is empty, an error code of 32 will be returned. In either case, the group will not be created.

Subsequent check of all the groups and their members created can be done by the methods **GetListNames** and **GetListEntries** (see Sections 7.1 and 7.2).

There is no method of adding extra members to an existing group. If adding members is desired, user can call **CreateGroup** again with the same group name and a complete list of members (including the extra members to be added).

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInGroupName	String	In	Name of the created group.
strInGroupMembers	String	In	A string of members of the group formatted as explained above.
iOutMembersCreated	Integer	Out	The returned number of members created for this group.

See also

GetListNames, GetListEntries

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
'---create a group called "TestGroup"
Dim strInGroupName As String = "TestGroup"
'---with 3 members
Dim strInGroupMembers As String =
  "Name1,Mobile1;Name2,Mobile2;Name3,Mobile3"
Dim iOutMembersCreated As Integer
.
.
Dim Result As Integer
Result = ws.CreateGroup(strInSerialNo,
                        strInSMSKey,
                        strInGroupName,
                        strInGroupMembers,
                        iOutMembersCreated)
.
.
```


7.2 GetListNames

Synopsis

```
int GetListNames(string strInSerialNo,
                 string strInSMSKey,
                 string &strOutListNames)
```

Description

This method returns the IDs, names and number of members of all the groups belonging to the user account in the output parameter **strOutListNames**.

The format of **strOutListNames** is as follows:

```
<Group Info 1>;...;<Group Info N>
```

Group information blocks are separated by "`;`". Each group information block comprises of 3 fields separated by "`;`" again, formatted as follows:

```
<Group ID>;<Group Name>;<Number of Members>
```

The group ID is an internally assigned ID for the group. This ID is returned so that user can use this ID to query members in this group by calling **GetListEntries** (see Section 7.3). Group name is the name specified in **strInGroupName** when calling **CreateGroup** (see Section 7.1). The last field is the number of members created for this group. If there is no group in the user account, the output parameter **strOutListNames** will be empty.

For example, suppose a user has 3 groups named "Group1", "Group2" and "Group3" with assigned IDs 101,102 and 103, and each group has 3 members. Then the following will be returned in **strOutListNames**:

```
101;Group1;3;102;Group2;3;103;Group3;3
```

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strOutListNames	String	Out	A string of groups information formatted as explained above.

See also

CreateGroup, GetListEntries

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strOutListNames As String  'output parameter
.
.
Dim Result As Integer
Result = ws.GetListNames(strInSerialNo,
                         strInSMSKey,
                         strOutListNames)
.
.
```

7.3 GetListEntries

Synopsis

```
int GetListEntries(string strInSerialNo,
                   string strInSMSKey,
                   int    iInListID,
                   string &strOutListEntries)
```

Description

This method returns the names and mobile numbers of all the members of a group in the output parameter **strOutListEntries**.

The group to be queried is specified by its ID in **iInListID**. This ID is obtained from calling **GetListNames** (see Section 7.2). The format of the output parameter **strOutListEntries** is as follows:

```
<Member Info 1>;<Member Info N>
```

Member information blocks are separated by "`;`". Each member information block comprises of 2 fields separated by "`;`" again, formatted as follows:

```
<Member Mobile>;<Member Name>
```

If any of the fields is empty (as allowed by **CreateGroup**), then the corresponding field position is also empty. If there is no member in the group, the output parameter **strOutListEntries** will be empty.

For example, suppose a user wants to query a group which has 3 members. The first and third members have names "Member1" and "Member3" respectively while the second member has no name. The members have mobile numbers "Mobile1", "Mobile2" and "Mobile3" respectively. Then the following will be returned in **strOutListEntries**:

```
Mobile1;Member1;Mobile2;;Mobile3;Member3
```


Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
iInListID	Integer	In	ID of the group to be queried.
strOutListEntries	String	Out	A string of members information of the group formatted as explained above.

See also

CreateGroup, GetListNames

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim iInListID As Integer = 101    'Group ID 101
Dim strOutListEntries As String   'output parameter
.
.
Dim Result As Integer
Result = ws.GetListNames(strInSerialNo,
                         strInSMSKey,
                         iInListID,
                         strOutListEntries)
.
.
```

8. Miscellaneous Methods

8.1 CheckMessageStatus

Synopsis

```
int CheckMessageStatus(string strInSerialNo,
                       string strInSMSKey,
                       string strInMessageIDs,
                       string &strOutMessageStatus)
```

Description

This method checks the status of previously deposited messages identified by the parameter **strInMessageIDs**.

The parameter **strInMessageIDs** is the output parameter **strOutMessageIDs** of the various '**Send…**' methods or **EnterScheduleExt** (see Section 5.2), that contains for each comma separated destination handset, a unique message ID. The format of this output parameter should be passed on, as is, to the input parameter **strInMessageIDs** of this method.

The current status of these messages will be returned in the parameter **strOutMessageStatus**. The format of the output is a comma separated list of message ID-status pairs:

```
messageID_1=status_1,...,messageID_n=status_n
```

Possible message status are:

Returned Status	Description
OK	The message has been dispatched to the SMSC successfully.
PENDING	The message is stored at the gateway waiting to be dispatched to the SMSC.
PROCESSING	The message is being processed, i.e. being dispatched to the SMSC. A message is only in this state for a very short time and should ultimately change to one of the other status so users will not see this status very often.
CONFIRMED	The message has been delivered to the destination handset successfully (if confirmation notification is enabled, see Section 2.4).

REPLIED	The message has been delivered to the destination handset successfully and the receiver has replied this message with the reply message sent to the reply email address of the sender. This only applies to two-way messages (see the method **SendTextSMS** in Section 3.1).
DOWNLOADED	This status pertains to MMS messages. It indicates that the MMS message has been retrieved or that a WAP Push version of it has been downloaded by the destination handset (see Chapter 4).
RETRY	The message fails to be dispatched to the SMSC, but the gateway will retry later.
FAIL_reason	The message has been dispatched to the SMSC successfully but a failure receipt is returned from the SMSC. The reason of the failure at the SMSC is included as “reason”.
ABORT	The message has experienced prolonged failure and is thus aborted.
FUTURE	The message is scheduled to be sent in the future. The date time of the schedule is however not shown. See the methods **EnterSchedule** and **EnterScheduleExt** (Sections 5.1 and 5.2).
CANCELLED / CANCEL	The message previously scheduled to be sent in the future by **EnterScheduleExt** has been cancelled by **DeleteSchedule** (see Section 5.3).
TESTING	The message is for testing purpose only and will not be dispatched.
INVALID	The message stored at the gateway was found to be invalid when the gateway is trying to dispatch it to the SMSC. This status mainly pertains to group sending when a user submits a message to a group (see Chapter 7). The group is expanded only when it is about to be sent, when the gateway may find out that the group is invalid, or one or more of the member destinations in the group are invalid.
NA	The message identifier is invalid or the message with the identifier is not found in the system database.

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strInMessageIDs	String	In	Comma separated list of message identifiers obtained from the various '**Send…**' methods or **EnterScheduleExt**, e.g. +4179123123:123456, +6140123123:123457
strOutMessageStatus	String	Out	Comma separated list of message ID-status pairs, e.g. 123456=OK,123457=PENDING

See also

The various '**Send…**' methods**, EnterSchedule, EnterScheduleExt, DeleteSchedule**

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strInOutMessageIDs As String
.
.
'---some Send... methods
'---or EnterScheduleExt are called
'---and the message identifiers
'---are stored in strInOutMessageIDs
.
.
Dim strOutMessageStatus As String
Dim Result As Integer
Result = ws.CheckMessageStatus(strInSerialNo,
                               strInSMSKey,
                               strInOutMessageIDs,
                               strOutMessageStatus)
.
.
```


8.2 GetCreditsLeft

Synopsis

```
int GetCreditsLeft(string strInSerialNo,
                   string strInSMSKey,
                   double &dOutCreditsLeft)
```

Description

This method returns how many credits or message quota are left for this user account in the output parameter **dOutCreditsLeft**.

Each SMS or MMS message that is sent using the various '**Send…**' methods will be charged for certain credits in the user account. It is sometimes useful to check how many credits are left. These credits are of type double since fractional credits could be charged, which is more flexible. However, they are usually integers (but formatted as double numbers).

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
dOutCreditsLeft	Double	Out	The returned number of credits or message quota left for this account, in double numbers.

Example

```
.
.
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim dOutCreditsLeft As Double
Dim Result As Integer
Result = ws.GetCreditsLeft(strInSerialNo,
                           strInSMSKey,
                           dOutCreditsLeft)
.
.
```

8.3 GetLicenseInformation

Synopsis

```
int GetLicenseInformation(string strInSerialNo,
                          string strInSMSKey,
                          string &strOutLicenseInfo)
```

Description

This method returns the information of the license of MIDA usage for the user in the output parameter **strOutLicenseInfo**. Note that MIDA allows a trial period of 30 days, after which, a fixed one-shot license payment is needed to continuing accessing MIDA.

The output parameter **strOutLicenseInfo** is a string that contains license information such as

- MIDA Version, currently it is 7.1.
- The name of the licensee and organization.
- If the license is an evaluation version (which normally has 30 days trial period), then the expiry date is given. The expiry date format is in mm/dd/yy.
- If the license is permanent, then the number of licenses (the number of licensed users) is given.
- The registered country.

Parameters

Parameters	Type	In/Out	Description
strInSerialNo	String	In	User serial number
strInSMSKey	String	In	User password (SMSKey)
strOutLicenseInfo	String	Out	The license information.

Example

```
Dim strInSerialNo As String = "my serial number"
Dim strInSMSKey As String = "my SMS key"
Dim strOutLicenseInfo As String
Dim Result As Integer
Result = ws.GetLicenseInformation(strInSerialNo, strInSMSKey,
                                  strOutLicenseInfo)
```


8.4 RedWebServiceVersion

Synopsis

```
string RedWebServiceVersion()
```

Description

This method simply returns the version of MIDA and has no parameters and error codes. The version is in the format:

```
<Major Version>.<Minor Version>
```

Currently, the version is 7.1.

Example

```
.
.
Dim Version As String
Version = ws.RedWebServiceVersion()
.
.
```

Appendix A: Error Codes

The return codes returned by the various methods signify success or failure of the corresponding access. The following table lists all the error codes and their descriptions (meanings, possible causes, etc.).

Error Code	Description
0	Success, no error
1	Feature not available. The requested service does not implement such feature, usually due to unsupported content type or incompatible content types and source types in sending binary SMS messages.
2	Service temporarily not available. Please try later. This code is superceded by code 16.
3	Too many previous invalid login attempts due to invalid passwords or serial numbers have been detected. For security reason, the account will be blocked for approximately 1 hour. The account will be unblocked again after this period and user can retry again.
4	Invalid password (SMS Key). For redcoal Communicator or EmailSMS user, the serial number may have expired and the user needs to re-login.
5	No more credits left. Please go to http://www.redcoal.com/purchase/ to purchase more credits.
6	Not enough credits left to complete service. Please go to http://www.redcoal.com/purchase/ to purchase more credits.
8	One or more invalid destinations (including empty destination) detected in the parameter **strInRecipients** of the various '**Send…**' methods.
10	Invalid serial number. Please contact us.

12	Daily quota reached. To stop users from sending too many messages, e.g. spamming, a certain daily quota limit is set on each account. If this quota is reached, no more messages can be sent (within this day). This quota is by default set to a high enough limit for average users and will be reset each day. The daily quota can also be set to an agreed limit for those customers that need large volume of messaging.
13	Destination not in restricted list. To prevent users from sending messages to unwanted destination, a list of valid destinations can be set up on an account basis. Messages sent to any destination not in this list are forbidden.
14	Invalid binary content format. When sending binary SMS messages using **SendBinarySMS**, certain contents expect certain source types, e.g. operator logo requires bitmap content, mono ring tone requires RTTTL format, etc. If the received binary content is not recognizable for a particular content type, then this error code will be returned. This error will also be returned if an invalid Base64-encoded string is received in the parameter **strInBase64Content** of **AddBase64MMSContent**.
15	Binary SM or MM too big. A maximum of 4 concatenated messages can be sent. Any contents require more than 4 messages will incur this error code.
16	General fault. Internal or gateway problems, e.g. no Internet connection, can't connect to gateway, can't get past the proxy firewall, etc.
18	Invalid license. The trial license has expired. Please go to http://www.redcoal.com/purchase/ to purchase license for continual access.
21	Invalid reply email address in the parameter **strInReplyEmail** of the various '**Send…**' methods.
31	Empty group name in the call to **CreateGroup**. Group will not be created.

32	Empty list of members in the call to **CreateGroup**. Group will not be created.
41	Transaction ID already exists. When creating MMS message using **CreateMMS**, a unique transaction ID, which can be user-supplied, is needed to identify it. If this ID is already used to identify another MMS message, it cannot be reused again and this error code will be returned.
42	Invalid transaction ID. When adding content parts to previously created MMS message using **AddMMSContent** or **AddBase64MMSContent**, or sending MMS message using **SendMMS**, the transaction ID is used to identify the MMS message previously created by **CreateMMS**. If this transaction ID cannot be found to match existing created MMS message, this error code will be returned.
43	Invalid content type. When adding content parts to previously created MMS message using **AddMMSContent** or **AddBase64MMSContent**, only certain content types are supported. Those content types that are not supported will incur this error code.
44	Empty MMS message content when calling **SendMMS**. No content part has been previously added to this MMS message using **AddMMSContent** or **AddBase64MMSContent**. At least one content part must be present and added before sending.
70	Invalid UTC format in the parameter **strInScheduledUTCDateTime** of the method **SendSMS2**.

Appendix B: Binary SM Content Types

The following table lists all supported binary SM content types and their descriptions. The table is in numeric order of the code. The codes for these content types are passed as the parameter **iInType** of the method **SendBinarySMS**, the parameter **iInContentType** of the method **SendBinarySMSByContent**, and in the parameter **iInOptions** of the method **SendSMS2**.

Content Type	Code	Description
Remove logo	255	This is used to send a special SM that will restore the original operator logo on the destination handset and the current one will effectively be removed. The SM only applies to Nokia's handsets.
EMS message	256	This content type requires the content source to be encoded in hex (see appendix C) and contains the EMS message. The EMS message can contain ring tones and simple animated text with sounds for EMS-enabled handsets by a variety of manufacturers, e.g. Ericsson. The EMS message can only be sent using **SendBinarySMSByContent**.
Motorola ring tone	848	This content type requires a Motorola proprietary ring tone. Motorola ring tone is actually a specially formatted text message that can be recognized by the Motorola's handsets as a ring tone. It contains a header part that begins with the string "L35" and the actual ring tone part. Since these ring tones are just texts, they are sent as text SMS messages and hence can be at most 160 characters long. Motorola ring tones longer than 160 characters are not supported.

Java Game	2960	This content type requires a java archive that contains the code for the java game. The java archive file (usually with file extension “jar”) must be used and the destination handsets must support java. In addition, since the java archive is downloaded to the handset using WAP, the handsets must also support WAP.
Mono Ring Tones	5505	This content type requires a mono ring tone. The ring tone file must be in RTTTL format (usually with file extension “rtttl”). The MIDA installation package provides some sample mono ring tone files. This content only applies to Nokia’s handsets. For sending mono ring tones to other handsets, see **SendBinarySMSByContent**.
Operator logo	5506	This content type requires an operator logo. The operator logo file is a 72x14 black and white bitmap file (usually with file extension “bmp”). When dispatching operator logos the 5 digits network operator code for destination network must be specified. Check out appendix D for valid network operator codes. The MIDA installation package provides some sample operator logo files. This content only applies to Nokia’s handsets.
Picture Message	5514	This content type requires a picture + (optional) text. The picture file is a 72x28 black and white bitmap file (usually with file extension “bmp”) and can be accompanied by up to 120 characters of text. The MIDA installation package provides some sample picture files. This content only applies to Nokia’s handsets.

WAP Bookmark	9039	This content type requires text that indicates some URL that can be reached using the WAP protocol from the destination handsets. For this reason, the handsets must support WAP. The maximum number of characters of the URL is 255 and the maximum number of characters of the name of the bookmark is 50. This content only applies to Nokia's handsets. For other handsets, the special WAP-push message called Service Indication should be used (see **SendWAPSI**).
VCard	9204	This content type requires a VCard. The VCard file must have the standard Microsoft VCard format (usually with file extension "vcf", but is actually a plain-text file in certain format). The MIDA installation package provides some sample VCard files. This content only applies to Nokia's handsets.
Polyphonic Ring Tones	9601	This content type requires a polyphonic ring tone. The ring tone file must be in MIDI format (usually with file extension "mid" or "midi"). It depends on the capability of the destination handsets of playing polyphonic sounds. The number of tracks stored in the MIDI file that can be played on the destination handset simultaneously also depends on the capability of the handsets. Check the manuals of the different makes and models of the handsets; most handsets can play a minimal of 4 tracks simultaneously. In addition, since the polyphonic ring tone is downloaded to the handsets using WAP, the handsets must also support WAP. The MIDA installation package provides some sample polyphonic ring tone files.

Appendix C: Binary SM Source Types

The method **SendBinarySMSByContent** accepts contents encoded in different source types, which compensates the fixed source types accepted by the method **SendBinarySMS** and allows more content types to be sent to a larger variety of handsets. These source types must be compatible with the intended content types.

The following table lists all supported source types, their codes, the compatible content type codes (see appendix B) and their descriptions. The table is in numeric order of the source codes. These source codes are passed as the parameter **iInSourceType** of the method **SendBinarySMSByContent**, or in the parameter **iInOptions** of the method **SendSMS2**.

Source Type	Code	Compatible Content Types	Description
NA	0	All	This source type does not specify any specific type and hence is not applicable (NA). When this is the case, **SendBinarySMSByContent** behaves just like **SendBinarySMS**; that is, the format of the input binary byte stream must conform to the corresponding format of the intended content accepted by **SendBinarySMS** (see appendix B).
Hex	1	All except: 848 2960 9601	This source type specifies a HEX encoded stream of the intended content. It is basically a plain text but the content is a stream of hex encoded strings, each two characters “XX” where X = 0..9,A..F, encode a binary byte. With this type, the users must encode the content appropriately themselves.
Plain text	2	848 9039 9204	This source type specifies plain text. The maximum number of characters must conform to the requirement of the intended content (see appendix B).
Bitmap	11	5506 5514	This source type specifies bitmaps. The format and size of the bitmaps must conform to the requirement of the intended content (see appendix B).

OTA	12	5506 5514	This source type specifies over-the-air (OTA) pictures.
RTTTL	21	5505	This source type specifies ring tones in RTTTL format.
MIDI	22	9601	This source type specifies polyphonic ring tones in MIDI format.
Java archive	23	2960	This source type specifies java archives that contain the codes for the java games.

Appendix D: Country and Network Operator Codes

Mobile Country Code (MCC) must be encoded as a 3-character string and Mobile Network Code (MNC) must be encoded as a 2-character string. The two codes must be separated by a single white space in between. For instance, the Australia Telstra code will be "505 01". This code is passed, as the parameter **strInExtraParam** of the method **SendBinarySMS**, **SendBinarySMSByContent** or **SendMobileContent**, or the parameter **strInContent2** of the method **SendSMS2**, when sending operator logo.

The following table lists the codes for all supported countries and network operators. The table is in alphabetical order of the countries. An up-to-date list can be found at http://www.redcoal.com/.

Country and Network Operator Name	Country and Network Operator Code
Albania Albanian Mobile Comms	276 01
Algeria Albanian Mobile Comms	603 01
Andorra S.T.A. MobilAnd	213 03
Armenia ArmenTel	283 01
Australia Telstra Mobile Comms	505 01
Australia Cable + Wireless Optus	505 02
Australia Vodafone	505 03
Austria MobilKom Austria A1	232 01
Austria max.mobil.Telekoms Service	232 03
Austria Connect Austria One	232 05
Azerbaijan Azercell Telekom B.M.	400 01
Azerbaijan J.V.Bakcell GSM 2000	400 02
Bahrain Batelco	426 01
Bangladesh Grameen Phone	470 01
Bangladesh Sheba Telecom	470 19
Belgium Belgacom Mobile Proximus	206 01
Belgium KPN Orange	206 20
Belgium Mobistar	206 10
Bosnia Herzegovina Cronet	218 01
Bosnia Herzegovina PTT Bosnia	218 19
Bosnia Herzegovina PE PTT BIH	218 90
Botswana Mascom Wireless	652 01
Brunei Darussalam Jabatan Telekom	528 01
Brunei Darussalam DST Communications	528 11
Bulgaria MobilTel AD	284 01
Cambodia CamGSM	456 01
Cambodia Cambodia Samart Comms	456 02
Cameroon PTT Cameroon Cellnet	624 01
Canada Microcell Connexions Inc	302 37
Cape Verde Cabo Verde Telecom	625 01
Chile Entel Telefonia Movil	730 01
Chile Entel PCS Telecom.	730 10
China China Telecom GSM	460 00
China China Unicom GSM	460 01
China Liaoning PPTA	460 02
Cote d'Ivoire Comstar Cellular Network	612 01
Cote d'Ivoire Telecel	612 02
Cote d'Ivoire S.I.M Ivoiris	612 03
Cote d'Ivoire Loteny Telecom Telecel	612 05
Croatia Croatian Telecoms Cronet	219 01
Croatia Vipnet	219 10
Cyprus Cyprus Telecoms Authority	280 01
Czech Republic RadioMobil	230 01
Czech Republic EuroTel Praha	230 02

Czech Republic SPT Telecom	230 03
Denmark Tele-Danmark Mobil	238 01
Denmark Sonofon	238 02
Denmark Telia Denmark	238 20
Denmark Mobilix	238 30
Egypt MobiNil	602 01
Egypt Misrfone Telecom. Click	602 02
Estonia Estonian Mobile Telephone	248 01
Estonia Radiolinja Eesti	248 02
Estonia Q GSM	248 03
Ethiopia Ethiopian Telecoms Auth.	636 01
Fiji Vodafone Fiji	542 01
Finland Telia Finland	244 03
Finland Radiolinja	244 05
Finland Alands Mobiltelefon	244 05
Finland Finnet Group	244 09
Finland Sonera Corporation	244 91
France France Telecom Itineris	208 01
France SFR	208 10
France Bouygues Telecom	208 20
French Polynesia Tikiphone	547 20
French West Indies France Caraibe Ameris	340 01
Georgia Geocell Limited	282 01
Georgia Magti GSM	282 02
Germany D1 DeTe Mobil	262 01
Germany D2 Mannesmann Mobilfunk	262 02
Germany E-Plus Mobilfunk	262 03
Germany Viag Interkom	262 07
Ghana ScanCom	620 01
Gibraltar Gibraltar Telecoms Gibtel	266 01
Greece Cosmote	202 01
Greece Panafon	202 05
Greece Telestet	202 10
Greenland Tele Greenland	290 01
Guinea Sotelgui Lagui	611 02
Hong Kong Hong Kong Telecom CSL	454 00
Hong Kong Hutchison Telecom	454 04
Hong Kong SmarTone Mobile Comms	454 06
Hong Kong New World PCS	454 10
Hong Kong Peoples Telephone	454 12
Hong Kong Mandarin Com. Sunday	454 16
Hong Kong Pacific Link	454 18
Hong Kong P Plus Comm	454 22
Hungary Pannon GSM	216 01
Hungary Westel 900 GSM Mobile	216 30
Iceland Iceland Telecom Siminn	274 01
Iceland TAL hf	274 02
India TATA Cellular	404 07
India Bharti Cellular Telecom Airtel	404 10
India Sterling Cellular Essar	404 11
India Escotel Mobile Comms	404 12
India Modi Telstra Modicom	404 14
India Aircel Digilink Essar Cellph.	404 15
India Hutchison Max Touch	404 20
India BPL Mobile	404 21
India BPL USWest Cellular	404 27
India Usha Martin Tel. Command	404 30
India Mobilenet	404 31
India SkyCell Communications	404 40
India RPG MAA	404 41
India Srinivas Cellcom	404 42
Indonesia PT. Satelindo	510 01
Indonesia Telkomsel	510 10
Indonesia PT. Excelcomindo Excelcom	510 11
Iran TCI	432 11
Iraq Iraq Telecom	418 01
Ireland Eircell	272 01

Ireland Esat Digifone	272 02
Ireland Meteor	272 03
Israel Partner Communications	425 01
Italy Telecom Italia Mobile TIM	222 01
Italy Omnitel Pronto	222 10
Italy Wind Telecomunicazioni	222 88
Jordan J.M.T.S Fastlink	416 01
Kuwait Mobile Telecoms MTCNet	419 02
Kyrgyz Republic Bitel	437 01
Lao Lao Shinawatra Telecom	457 01
Latvia Latvian Mobile Tel.	247 01
Latvia BALTCOM GSM	247 02
Lebanon FTML Cellis	415 01
Lebanon LibanCell	415 03
Lesotho Vodacom	651 01
Liberia Omega Communications	618 01
Lithuania Omnitel	246 01
Lithuania UAB Bite GSM	246 02
Luxembourg P+T LUXGSM	270 01
Luxembourg Millicom Tango GSM	270 77
Macau C.T.M. TELEMOVEL+	455 01
Macedonia Macedonian Tel. MobiMak	294 01
Madagascar Madacom	646 01
Madagascar SMM Antaris	646 02
Madagascar Sacel	646 03
Malawi Telekom Network Callpoint	650 01
Malaysia My BSB	502 02
Malaysia Binariang	502 03
Malaysia Binariang Comms. Maxis	502 12
Malaysia Telekom Cellular TM Touch	502 13
Malaysia DiGi Telecommunications	502 16
Malaysia Time Wireless Adam	502 17
Malaysia Celcom	502 19
Malta Vodafone	278 01
Mauritius Cellplus Mobile Comms	617 01
Moldova Voxtel	259 01
Morocco Itissalat Al-Maghrib IAM	604 01
Mozambique Telecom de Mocambique	634 01
Namibia MTC	649 01
Netherlands Libertel	204 04
Netherlands KPN Telecom	204 08
Netherlands Telfort	204 12
Netherlands Ben	204 16
Netherlands Dutchtone	204 20
New Caledonia OPT Mobilis	546 01
New Zealand Vodafone	530 01
New Zealand Telecom NZ	530 03
New Zealand Telstra	530 04
Norway Telenor Mobil	242 01
Norway NetCom GSM	242 02
Oman General Telecoms	422 02
Pakistan Mobilink	410 01
Papua New Guinea Pacific Mobile Comms	310 01
Philippines Isla Comms	515 01
Philippines Globe Telecom	515 02
Philippines Smart Communications	515 03
Poland Polkomtel PLUS GSM	260 01
Poland ERA GSM	260 02
Poland IDEA Centertel	260 03
Portugal Telecel Communicacoes	268 01
Portugal Optimus Telecom.	268 03
Portugal Telecom Moveis Nac. TMN	268 06
Qatar Q-Tel QATARNET	427 01
Reunion Societe Reunionnaise SRR	647 10
Romania MobiFon CONNEX GSM	226 01
Romania Mobil Rom DIALOG	226 10
Russia MTS Moscow	250 01

Russia North-West GSM	250 02
Russia Siberian Cellular	250 05
Russia Zao Smarts	250 07
Russia Don Telecom	250 10
Russia New Telephone Company	250 12
Russia Far-Eastern Cellular	250 12
Russia Kuban GSM	250 13
Russia Uratel	250 39
Russia North Caucasian GSM	250 44
Russia KB Impuls BeeLine	250 99
Rwanda Rwandacell	635 10
Saudi Arabia Ministry of PTT Al Jawal	420 01
Saudi Arabia Electronics App' Est. EAE	420 07
Senegal Sonatel ALIZE	608 01
Seychelles Seychelles Cellular Services	633 01
Seychelles Telecom AIRTEL	633 10
Singapore Singapore Tel. GSM 900	525 01
Singapore Singapore Tel. GSM 1800	525 02
Singapore MobileOne Asia	525 03
Slovak Republic Globtel GSM	231 01
Slovak Republic EuroTel GSM	231 02
Slovenia Si.mobil	293 40
Slovenia Mobitel	293 41
South Africa Vodacom	655 01
South Africa MTN	655 10
Spain Airtel Movil	214 01
Spain Retevision Movil	214 03
Spain Telefonica Moviles Movistar	214 07
Sri Lanka MTN Networks Dialog GSM	413 02
Sudan Mobile Telephone Company	634 01
Sweden Telia Mobitel	240 01
Sweden Comviq GSM	240 07
Sweden Europolitan	240 08
Switzerland Swisscom NATEL	228 01
Switzerland diAx Mobile	228 02
Switzerland Orange	228 03
Syria Syrian Telecom Est. MOBILE	417 09
Taiwan Far EasTone Telecoms	466 01
Taiwan TUNTEX Telecom	466 06
Taiwan KG Telecom	466 88
Taiwan Chunghwa Telecom	466 92
Taiwan Mobitai Communications	466 93
Taiwan Pacific Cellular TWNGSM	466 97
Taiwan TransAsia Telecoms	466 99
Tanzania Tritel	640 01
Thailand Advanced Info Service AIS	520 01
Thailand WCS IQ	520 10
Thailand Total Access Worldphone	520 18
Thailand Digital Phone HELLO	520 23
Togo Togo Telecom TOGO CELL	615 01
Tunisia Tunisie Telecom Tunicell	605 02
Turkey Turk Telekom Turkcell	286 01
Turkey TELSIM Mobil Telekom.	286 02
U.S.A. APC Sprint Spectrum	310 02
U.S.A. Wireless 2000 Telephone	310 11
U.S.A. BellSouth Mobility DCS	310 15
U.S.A. Omnipoint Communications	310 16
U.S.A. Pacific Bell Wireless	310 17
U.S.A. Western Wireless Voicestream	310 26
U.S.A. Powertel	310 27
U.S.A. Aerial Communications	310 31
U.S.A. Iowa Wireless Services	310 77
Uganda Celtel Cellular	641 01
Uganda MTN Uganda	641 10
Ukraine Ukrainian Mobile Comms	255 01
Ukraine Ukrainian Radio Systems	255 02
Ukraine Kyivstar GSM	255 03

Ukraine Golden Telecom	255 05
United Arab Emirates UAE ETISALAT-G1	424 01
United Arab Emirates UAE ETISALAT-G2	424 02
United Kingdom Cellnet	234 10
United Kingdom Vodafone	234 15
United Kingdom One 2 One	234 30
United Kingdom Orange	234 33
United Kingdom Jersey Telecom GSM	234 50
United Kingdom Guernsey Telecoms GSM	234 55
United Kingdom Manx Telecom Pronto GSM	234 58
Uzbekistan Buztel	434 01
Uzbekistan Daewoo Unitel	434 04
Uzbekistan Coscom	434 05
Venezuela Infonet	734 01
Vietnam MTSC	452 01
Vietnam DGPT	452 02
Yugoslavia MOBTEL	220 01
Yugoslavia ProMonte GSM	220 02
Zambia Zamcell	645 01
Zimbabwe NET*ONE	648 01
Zimbabwe Telecel	648 03

– End of Document –