Send SMS via SOAP API

Introduction

You can seamlessly integrate your applications with aql's outbound SMS messaging service via SOAP using our SOAP API.

Sending messages via the SOAP gateway

WSDL file

In order to create your proxy classes, we have provided a WSDL file. This can be located at the following URL:

http://gw.aql.com/soap/sendservice.php?WSDL

There are four operations permitted within the SendSmsService service (as defined in the WSDL file above). These are:

Operation Name	Description
SoapSendSms	Use this operation to send a normal or flash SMS message
SoapSendBinarySms	Use this operation to send a binary SMS message. You must supply the correct UDH and DATA elements
SoapSendWapPush	Use this operation to send a WAP Push message. You must supply the WAP title and WAP URL
SoapSendMMSNot	Use this operation to send an MMS notification message

Operation specifications

This section gives a description of each of the operations listed in the table above. This includes the various element names, their type, list of possible values (if applicable, e.g. a restricted type) if it is optional and examples of the elements use.

For any elements that have restricted types (i.e. they can only have certain values), the permitted values will be listed in single quotes. An example of this is the **messagetype** element in the SoapSendSms operation.

All types prefixed with **xs:** are from the http://www.w3.org/2001/XMLSchema namespace. Information on the types available within the namespace can be found here.

SoapSendSms

Parameter: destinationarray

The destinationarray element is used to supply us with the destination numbers. These must be in international format without the leading '+' and without any leading zeros. Please see the example code for a reference on how to use this element

Type: Array of number elements

any valid MSISDN

Optional: No

Code example:

```
<destinationarray>
     <number>447766404142</number>
     <number>447777777111</number>
     <number>447749494949</number>
</destinationarray>
```

Parameter: originator

The originator sets who the message appears to be from. This can be any MSISDN (up to 16 digits) or any alphanumeric string (up to 11 characters). If you supply an alphanumeric originator, it best to avoid using any symbols (non-A-Z 0-9 characters) to ensure maximum compatibility between different handsets. Please see the example code for a reference on how to use this element

Type: xs:string

Restricted strings

When sending SMS messages, we recommend that certain strings have restrictions placed upon them. This includes the originator, the WAP title in a WAP push message and the subject in an MMS notification. This is to ensure maximum compatibility between different handsets. Our gateway does not enforce any of these restrictions, as there are handsets that can accept messages with characters from the whole character set.

If you are not sure what type of handsets you are messaging to or if they can handle the full character sets in these fields, you are probably best to restrict the characters to the following:

A-Z, a-z, 0-9, - (hyphen), _ (underscore)

Optional: Yes

Code example:

```
<originator>447766404142</originator>

or

<originator>alphastring</originator>
```

Parameter: message

The message element contains that actual message you would like to send to the destinations.

Type: xs:string

GSM character set

When a normal SMS message is sent to a handset, it is always converted into the GSM character set. There are certain characters which, when converted to the GSM character set, actually take up the space of two characters instead of one. These characters are:

^, {, }, \, [,], ~, |,
0x0C (Form feed in ASCII),
0x80 (Euro char in GSM. Non-graphic char in ISO-8859-1)

This is best illustrated by an example. The following string takes up 16 characters (excluding the quotes):

'This is a string'

The following string now takes up 20 GSM characters as the brackets each take up two characters instead of one:

'This is a [string]'

The number of characters that are allowed in each message are 160. The GSM character set issue basically means that this limit could be reduced to 159 or less depending on how many of the extended characters you place within your message. i.e., if your messages were to consist entirely of the backslash character '\', you would only be able to fit 80 of these per message.

Optional: No

Code example:

```
<message>your message here</message>
```

Parameter: messagetype

The messagetype element is used to define what type of SMS message you would like to send. Use 'text' for a normal SMS or use 'flash' to send a flash message. Flash messages are SMS messages that are immediately displayed on the screen. They are also usually not saved in the inbox. Flash messages are usually handset dependant, so it would be advisable to send a test message to handset of the same type before sending any flash messages.

Type: xs:string

'text',
'flash'

Optional: No

Code example:

```
<messagetype>flash</messagetype>

or

<messagetype>text</messagetype>
```

Parameter: maxconcat

your message goes over 160 characters, it will be split over more than one message. This value determines the maximum number of SMS messages that will be used to transmit the message. If this element is omitted, the default value on your account will be used.

Type: xs:integer

1-255

Optional: Yes

Code example:

```
<maxconcat>2</maxconcat>
```

Parameter: sendtime

This optional element can be used to specify a time in the future to send the message(s). The date format is YYYY-MM-DDTHH:MM:SSZ. The 'T' and 'Z' characters are literals and need to be in the date. The time zone is not supported - if supplied, it will be ignored.

Type: xs:dateTime

any valid date

Optional: Yes

Code example:

```
<sendtime>
     2005-10-10T17:00:00Z
</sendtime>
```

Parameters: dlrep

This optional element can be used to notify you of delivery/failure of messages. You can specify the callback URL using the callbackurl element. You must also set the callbacktype element to 'HTTPPOST'.

Type: callbackelement

see callback type

Optional: Yes

Code example:

```
<dlrep>
     <callbackurl>http://your.url</callbackurl>
<callbacktype>HTTPGET</callbacktype>
</dlrep>
```

SoapSendWapPush

Parameter: destinationarray

The destinationarray element is used to supply us with the destination numbers. These must be in international format without the leading '+' and without any leading zeros. Please see the example code for a reference on how to use this element.

Type: Array of number elements

any valid MSISDN

Optional: No

Code example:

```
<destinationarray>
     <number>447766404142</number>
```

```
    <number>447777777111</number>
    <number>447749494949</number>
</destinationarray>
```

Parameter: originator

The originator sets who the message appears to be from. This can be any MSISDN (up to 16 digits) or any alphanumeric string (up to 11 characters). If you supply an alphanumeric originator, it best to avoid using any symbols (non-A-Z 0-9 characters) to ensure maximum compatibility between different handsets. Please see the example code for a reference on how to use this element

Type: xs:string

Restricted strings

When sending SMS messages, we recommend that certain strings have restrictions placed upon them. This includes the originator, the WAP title in a WAP push message and the subject in an MMS notification. This is to ensure maximum compatibility between different handsets. Our gateway does not enforce any of these restrictions, as there are handsets that can accept messages with characters from the whole character set.

If you are not sure what type of handsets you are messaging to or if they can handle the full character sets in these fields, you are probably best to restrict the characters to the following:

A-Z, a-z, 0-9, - (hyphen), _ (underscore)

Optional: Yes

Code example:

```
<originator>447766404142</originator>
or
<originator>alphastring</originator>
```

Parameter: waptitle

This element is used to specify a title that is displayed on the destination handset

Type: xs:string

Restricted strings

When sending SMS messages, we recommend that certain strings have restrictions placed upon them. This includes the originator, the WAP title in a WAP push message and the subject in an MMS notification. This is to ensure maximum compatibility between different handsets. Our gateway does not enforce any of these restrictions, as there are handsets that can accept messages with characters from the whole character set.

If you are not sure what type of handsets you are messaging to or if they can handle the full character sets in these fields, you are probably best to restrict the characters to the following:

A-Z, a-z, 0-9, - (hyphen), _ (underscore)

Optional: No

Code example:

```
<waptitle>a wap title</waptitle>
```

Parameter: wapurl

This element is used to specify the WAP URL. A WAP push can sometimes be split over more than 1 message. This depends upon the length of the WAP URL. To keep the credit usage low, it is best to have a shorter URL.

Type: xs:string

any valid URL

Optional: No

Code example:

```
<wapurl>http://some.wap.url</wapurl>
```

Parameter: sendtime

This optional element can be used to specify a time in the future to send the message(s). The date format is YYYY-MM-DDTHH:MM:SSZ. The 'T' and 'Z' characters are literals and need to be in the date. The time zone is not supported - if supplied, it will be ignored.

Type : xs:dateTime

any valid date

Optional: Yes

Code example:

```
<sendtime>

     2005-10-10T17:00:00Z

</sendtime>
```

dlrep

This optional element can be used to notify you of delivery/failure of messages. You can specify the callback url using the callbackurl element. You must also set the callbacktype element to 'HTTPPOST'.

Type: callbackelement

Optional: Yes

Code example:

```
<dlrep>

     <callbackurl>http://your.url</callbackurl>
<callbacktype>HTTPGET</callbacktype>

</dlrep>
```

SoapSendBinarySms

Parameter: destinationarray

The destinationarray element is used to supply us with the destination numbers. These must be in international format without the leading '+' and without any leading zeros. Please see the example code for a reference on how to use this element.

Type: Array of number elements

any valid MSISDN

Optional: No

Code example:

```
<destinationarray>

     <number>447766404142</number>

     <number>447777777111</number>

     <number>447749494949</number>

</destinationarray>
```

Parameter: originator

The originator sets who the message appears to be from. This can be any MSISDN (up to 16 digits) or any alphanumeric string (up to 11 characters). If you supply an alphanumeric originator, it best to avoid using any symbols (non-A-Z 0-9 characters) to ensure maximum compatibility between different handsets. Please see the example code for a reference on how to use this element.

Type: xs:string

Restricted strings

When sending SMS messages, we recommend that certain strings have restrictions placed upon them. This includes the originator, the WAP title in a WAP push message and the subject in an MMS notification. This is to ensure maximum compatibility between different handsets. Our gateway does not enforce any of these restrictions, as there are handsets that can accept messages with characters from the whole character set.

If you are not sure what type of handsets you are messaging to or if they can handle the full character sets in these fields, you are probably best to restrict the characters to the following:

A-Z, a-z, 0-9, - (hyphen), _ (underscore)

Optional: Yes

Code example:

```
<originator>447766404142</originator>

or

<originator>alphastring</originator>
```

Parameter: data

This element should contain the data part of the binary SMS.

Type: xs:hexBinary

Restricted strings

When sending SMS messages, we recommend that certain strings have restrictions placed upon them. This includes the originator, the WAP title in a WAP push message and the

subject in an MMS notification. This is to ensure maximum compatibility between different handsets. Our gateway does not enforce any of these restrictions, as there are handsets that can accept messages with characters from the whole character set.

If you are not sure what type of handsets you are messaging to or if they can handle the full character sets in these fields, you are probably best to restrict the characters to the following:

A-Z, a-z, 0-9, - (hyphen), _ (underscore)

Optional: No

Code example:

```
<data>
      01060403AE81EA02056A0045C60b03687474703a
      2f2f7777772e61716c2e636f6d2f7761702f6171
      6c2e776d6c001103314061716c00080103457861
      6d706c652057415020707573682066726f6d2061
      716c000101
</data>
```

Parameter: udh

This element should contain the UDH part of the binary SMS.

Type: xs:hexBinary

0-9A-Z

Optional: No

Code example:

```
<udh>
      0605040B8423F0
</udh>
```

Parameter: sendtime

This optional element can be used to specify a time in the future to send the message(s). The date format is YYYY-MM-DDTHH:MM:SSZ. The 'T' and 'Z' characters are literals and need to be in the date. The time zone is not supported - if supplied, it will be ignored.

Type: xs:dateTime

any valid date

Optional: Yes

Code example:

```
<sendtime>

     2005-10-10T17:00:00Z

</sendtime>
```

Parameter: dlrep

This optional element can be used to notify you of delivery/failure of messages. You can specify the callback url using the callbackurl element. You must also set the callbacktype element to 'HTTPPOST'.

Type: callbackelement

Optional: Yes

Code example:

```
<dlrep>

     <callbackurl>http://your.url</callbackurl>
<callbacktype>HTTPGET</callbacktype>

</dlrep>
```

Parameter: encoding

A binary SMS can be sent with an encoding of either 7, 8 or 16 bit. For example, A concatenated SMS uses 7 bit, a WAP push uses 8 bit and Unicode messaging uses 16 bit.

Type: xs:integer

'7',
'8',
'16'

Optional: Yes

Code example:

```
<encoding>7</encoding>
```

SoapSendMMSNot

Parameter: destinationarray

The destinationarray element is used to supply us with the destination numbers. These must be in international format without the leading '+' and without any leading zeros. Please see the example code for a reference on how to use this element

Type: Array of number elements

any valid MSISDN

Optional: No

Code example:

```
<destinationarray>
     <number>447766404142</number>
     <number>447777777111</number>
     <number>447749494949</number>
</destinationarray>
```

Parameter: originator

The originator sets who the message appears to be from. This can be any MSISDN (up to 16 digits) or any alphanumeric string (up to 11 characters). If you supply an alphanumeric originator, it best to avoid using any symbols (non-A-Z 0-9 characters) to ensure maximum compatibility between different handsets. Please see the example code for a reference on how to use this element.

Type: xs:string

Restricted strings

When sending SMS messages, we recommend that certain strings have restrictions placed upon them. This includes the originator, the WAP title in a WAP push message and the subject in an MMS notification. This is to ensure maximum compatibility between different handsets. Our gateway does not enforce any of these restrictions, as there are handsets that can accept messages with characters from the whole character set.

If you are not sure what type of handsets you are messaging to or if they can handle the full character sets in these fields, you are probably best to restrict the characters to the following:

A-Z, a-z, 0-9, - (hyphen), _ (underscore)

Optional: Yes

Code example:

```
<originator>447766404142</originator>

or

<originator>alphastring</originator>
```

Parameter: subject

This is the subject for the notification message. It is what will be displayed in the inbox for example.

Type: xs:string

Restricted strings

When sending SMS messages, we recommend that certain strings have restrictions placed upon them. This includes the originator, the WAP title in a WAP push message and the subject in an MMS notification. This is to ensure maximum compatibility between different handsets. Our gateway does not enforce any of these restrictions, as there are handsets that can accept messages with characters from the whole character set.

If you are not sure what type of handsets you are messaging to or if they can handle the full character sets in these fields, you are probably best to restrict the characters to the following:

A-Z, a-z, 0-9, - (hyphen), _ (underscore)

Optional: Yes

Code example:

```
<subject>a subject</subject>
```

Parameter: mmsurl

This field contains the fully qualified URL to the mms file. It can affect the number of SMS credits used to transmit this message. The longer the URL, the higher the chance of it needing to be split over two or more messages.

Type : xs:string

any valid URL

Optional: Yes

Code example:

```
<mmsurl>http://domain.com/mmsfile.mms</mmsurl>
```

Parameter: filesize

This field must contain the size of the MMS file in bytes.

Type: xs:integer
Positive integer
Optional: Yes
Code example:

```
<filesize>2073</filesize>
```

Parameter: class

There are various classes an MMS notification be in. These are Personal (80), Advertisement (81), Informational (82), Auto (83). You must provide one of the integers in this field.

Type: xs:integer
'80',
'81',
'82',
'83'
Optional: No
Code example:

```
<class>80</class>
```

Parameter: expiry

This field contain the number of days after which this message is to expire.

Type: xs:integer
Positive integer. (Days)
Optional: Yes
Code example:

```
<expiry>4</expiry>
```

Parameter: sendtime

This optional element can be used to specify a time in the future to send the message(s). The date format is YYYY-MM-DDTHH:MM:SSZ. The 'T' and 'Z' characters are literals and need to be in the date. The time zone is not supported - if supplied, it will be ignored.

Type: xs:dateTime

any valid date

Optional: Yes

Code example:

```
<sendtime>

     2005-10-10T17:00:00Z

</sendtime>
```

Parameter: dlrep

This optional element can be used to notify you of delivery/failure of messages. You can specify the callback url using the callbackurl element. You must also set the callbacktype element to 'HTTPPOST'.

Type: callbackelement

Optional: Yes

Code example:

```
<dlrep>

     <callbackurl>http://your.url</callbackurl>
<callbacktype>HTTPGET</callbacktype>

</dlrep>
```

aql subtypes

callback type

The callbackelement type (which the dlrep is a type of) consists of two sub-elements. These are the type of callback and the callback URL:

callbackelement

Parameter: callbackurl

The callbackurl needs to be set to the script our servers will call to send you a delivery notification.

Type: xs:string

any valid URL

Optional: No

Code example:

```
<callbackurl>http://your.url.com/a.php</callbackurl>
```

Parameter: callbacktype

This needs to be set to either HTTPGET or NONE. If 'NONE' is specified, our system will ignore the callbackurl. It would be identical to omitting the dlrep element from the main SOAP message

Type: xs:string

'HTTPGET',
'NONE'

Optional: No

Code example:

```
<callbacktype>HTTPGET</callbacktype>
```

Delivery Notification via HTTP

An example of a callback URL is:

http://yourdomain.com/yourscript.php?reportcode=%code&destinationnumber=%dest&myreference=123

You must provide the '**%code**' and the '**%dest**' literals in your callback URL because these are replaced with the destination number and the resultcode.

The resultcode determines the status of the message at that particular time. The possible values of this callback are:

1 = Delivered to Handset
2 = Rejected from Handset
4 = Buffered in transit (phone probably off / out of reception)
8 = Accepted by SMSC
16 = Rejected by SMSC

You can set any other variables in the callback for your own tracking; for example, **myreference** in this case.

Note: If the message you send has to be split into multiple parts (e.g. a WAP push or a concatenated message), you will receive a delivery notification for each part.

Sample code

For sample code on how to use our SOAP interface in ASP.NET - C# (C Sharp): SMS SOAP Sample Code.

Note: You would need to compile the SOAP library from the WSDL.

```
/**
 * Sample code on how to send sms via soap using C#
 *
 * Copyright (c) 2007, aq Limited
 * All rights reserved.
 */


using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using soapFunction.com.aql.gw1;
using System.Web.Services.Protocols;


namespace soapFunction
{
  // Summary description for Form1.
  public class Form1 : System.Windows.Forms.Form
  {
    private System.Windows.Forms.Button button1;


    // Required designer variable.
    private System.ComponentModel.Container components = null;


    private SendSmsService cpLatestBrief = null;


    public Form1()
    {
      // Required for Windows Form Designer support
      InitializeComponent();


      cpLatestBrief = new SendSmsService();
    }
```

```
// Clean up any resources being used.
protected override void Dispose( bool disposing )
{
  if( disposing )
  {
    if (components != null)
      components.Dispose();
  }
  base.Dispose( disposing );
}



# region Windows Form Designer generated code
// Required method for Designer support - do not modify
// the contents of this method with the code editor.
private void InitializeComponent()
{
  this.button1 = new System.Windows.Forms.Button();
  this.SuspendLayout();


  // button1
  this.button1.Location = new System.Drawing.Point(80, 56);
  this.button1.Name = "button1";
  this.button1.Size = new System.Drawing.Size(104, 32);
  this.button1.TabIndex = 0;
  this.button1.Text = "button1";
  this.button1.Click += new System.EventHandler(this.button1_Click);


  // Form1
  this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
  this.ClientSize = new System.Drawing.Size(292, 266);
  this.Controls.Add(this.button1);
  this.Name = "Form1";
  this.Text = "Form1";
  this.ResumeLayout(false);
}


# endregion


/// The main entry point for the application.


[STAThread]
static void Main()
{
  Application.Run(new Form1());
}


public void sendansms()
{
```

```
        // Message details
        string originator = "sender";
        string message = "test message 1";
        string [] destarr = new string[1];
        destarr[0] = "447123456789"; // destination number


        // create authentication object
        cpLatestBrief.authValue = new auth();
        cpLatestBrief.authValue.username = "username"; // your aql username
        cpLatestBrief.authValue.password = "password"; // your aql password
        string creditsused;
        string outdesr;


        System.DateTime dt = DateTime.Now;
        com.aql.gw1.callbackelement cbe = new callbackelement();
        cbe.callbackurl = "http://url.com";
        cbe.callbacktype = com.aql.gw1.callbacktypeoptions.HTTPGET;


        if (cpLatestBrief != null)
        {
          try
          {

cpLatestBrief.SoapSendSms(destarr,originator,message,com.aql.gw1.mtype.text
,"2",dt,false,cbe,out creditsused,out outdesr);
            MessageBox.Show(creditsused, "Message sent, 1 credit is
deducted",MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
          }
          catch(SoapHeaderException e)
          {
            MessageBox.Show(e.Message, "Name Entry
Error",MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
          }
          catch(SoapException e)
          {
            System.Xml.XmlNode node = e.Detail;
            string errorcode = e.Detail.FirstChild.InnerText;
            string errortext = e.Detail.FirstChild.NextSibling.InnerText;


            if(errorcode == "DESTERROR-INVALIDNUMBERS")
              MessageBox.Show(e.Message, "Name Entry
Error",MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            else if(errorcode == "MESSAGE-NOTSUPPLIED")
              MessageBox.Show(e.Message, "Name Entry
Error",MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            else
              MessageBox.Show(errorcode+e.Message+"  "+errortext, "Name Entry
Error",MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
          }
          return;
        }
        else
          return;
      }
```

```
    private void button1_Click(object sender, System.EventArgs e)
    {
      sendansms();
    }
  }
}
```

Important considerations

Restricted strings

When sending SMS messages, we recommend that certain strings have restrictions placed upon them. This includes the originator, the WAP title in a WAP push message and the subject in an MMS notification. This is to ensure maximum compatibility between different handsets. Our gateway does not enforce any of these restrictions, as there are handsets that can accept messages with characters from the whole character set.

If you are not sure what type of handsets you are messaging to or if they can handle the full character sets in these fields, you are probably best to restrict the characters to the following:

A-Z, a-z, 0-9, - (hyphen), _ (underscore)

GSM character set

When a normal SMS message is sent to a handset, it is always converted into the GSM character set. There are certain characters, when converted to the GSM character set, actually take up the space of two characters instead of one. These characters are:

^, {, }, \, [,], ~, |,
0x0C (Form feed in ASCII),
0x80 (Euro char in GSM. Non-graphic char in ISO-8859-1)

This is best illustrated by an example. The following string takes up 16 characters (excluding the quotes):

'This is a string'

The following string now takes up 20 GSM characters as the brackets each take up two characters instead of one:

'This is a [string]'

The number of characters that are allowed in each message are 160. The GSM character set issue basically means that this limit could be reduced to 159 or less depending on how many of the extended characters you place within your message. i.e., if your messages were to

consist entirely of the backslash character '\', you would only be able to fit 80 of these per message.

If you have questions on this or any other subject, please raise a support query. Our staff will then respond to your query with an average response time of around 15 minutes.