DRAGONFLY SMS
DEVELOPERS GUIDE

SMS WEB SERVICE

CONTENTS

CHANGE HISTORY

Version	Date	Author	Notes
0.61	26/08/2016	Sal Javed	Initial draft for review
0.62	19/09/2016	Greg Neilson	Editing of draft
0.63	22/09/2016	Sal Javed	Update post review with further intro/section changes
0.64	28/09/2016	Greg Neilson	Post review
0.65	13/10/2016	Sal Javed	Final adjustment
1	31/11/2016	Greg Neilson	Samples added for C#, VB.NET, PHP and Java

DEVELOPERS GUIDE

INTRODUCTION

Dragonfly is the developer brand used by M:Science Ltd. M:Science are a market leader in SMS text message aggregation and software with over 15 years' experience and trading. M:Science aggregate millions of message every year for its corporate clients and Dragonfly exists to simplify the use of our tools within the developer community.

The SMS Web Service is a unique SOAP API served using HTTP and HTTPS, which allows organisations to integrate two-way SMS messaging functionality into their own websites, applications or web services.

Messages can be sent to handsets (MT, or Mobile Terminated, messages) or received from handsets (MO, or Mobile Originated, messages) to special numbers rented from M Science.

MO messages sent to numbers rented from M Science can be routed in a variety of methods, i.e. MO messages can be sent to the M Science SMS product, via Web PUSH or via calls to the Web Service. It is possible to specify a different method for each rented number. This document concerns itself with MO messages retrieved from our web service via calls to the API. As the product is .NET enabled and access to the web service uses standard Internet protocols, it is vendor, platform and language independent enabling multiple methods of connection.

The description of the SMS Web Service can be seen on the WSDL document that stands for Web Services Description Language, which is to describe web services, written in XML and has been W3C recommendation since 26th June 2007.

To view the WSDL document for the web service, visit the following URL:

http://smswebservice1.m-science.com/MScienceSMSWebService.asmx?WSDL

This allows external users to communicate with the service via HTTP/S or SOAP to send MT messages and view delivery receipts, or configure asynchronous routing of MO messages. This is done using SOAP method calls or HTTP GET or PUSH operations onto the Web service.

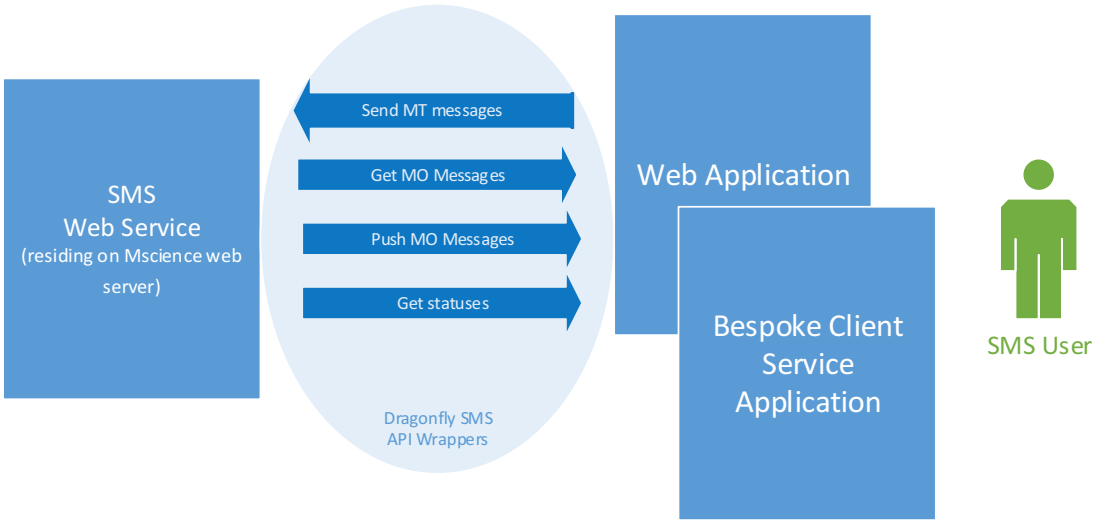
THE SMS Wrapper API has been developed to better surface the methods within the web service to code written in many different programming languages

Secure communications via HTTPS are also supported.

SMS API OVERVIEW

The API uses SOAP calls over the HTTP protocol. Languages supported are C#, VB.net, ASP.net, Java, PHP and Python If your favoured language is not supported, please contact M Science and we'll look at adding support.

The API allows external users to use the Web Service for MO/MT messaging and/or use the Web Push Service for allowing MO messages to use the 'PUSH' mechanism send messages to your web service when they are received to our MScience servers.



Web Service

Hosted by M:Science, this .NET web service allows external users to communicate with the service via HTTP or SOAP to send, receive and monitor text messages or configure asynchronous inbound routing of text messages. This is done using SOAP method calls or HTTP GET or PUSH operations onto the Web service SMS Web Service server.

Web Push Service

This service asynchronously routes MO messages via HTTP PUSH to one or more registered URLs. A web address must be provided which interprets the parameter list specified and allows a return response to indicate successful transmission of the payload.

SENDING MESSAGES

SENDING ONE OR MORE MESSAGES

Messages are sent by making calls with 'Send', which will allow up to five messages to be supplied. A message ID is then returned for each message which can be used to track the progress of the message.

TRACKING MESSAGES

The status of a message can be obtained by making a call onto 'GetMessageStatus', which will accept up to ten message IDs and then return a status for each one.

To fully track a message, it is recommended to request a delivery receipt when the original message is sent. If the client needs to interpret delivery receipts directly, the 'Source ID' field can also be supplied with the original message. This is sent back with any delivery receipt and can hence be used to tie the two together.

RECEIVING MESSAGES BY POLLING

If a selected inbound number has been configured to return messages to the web service, the messages can be retrieved by making calls onto 'GetInboundMessages'. This will return up to the next five messages in the queue.

RECEIVING MESSAGES BY ASYNCHRONOUS PUSH

It is possible to configure the M:Science SMS Web Service server to asynchronously 'PUSH' messages to a web address, calling the local software immediately the message arrives. The user must first contact M:Science at enquires@m-science.com to activate their account for this purpose and then must register one or more URLs using 'RegisterInbound'. The web server code behind the URLs must be capable of accepting an HTTP PUSH (the same as a form submit) using the parameters specified in the 'Client Interface section of this guide.

To stop message pushing, either a specific URL can be unregistered using 'RemoveInbound' .

The client can check to see if the web service is having problems pushing messages by calling onto 'GetInboundStatus' and stating the URL.

SMSCLIENT CLASS

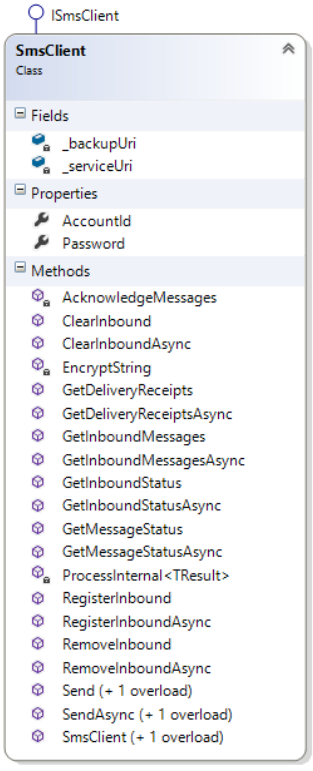
The fundamental class to use is ***SmsClient*** which has exposed methods to send/receive allow the status of messages for the Web Service, and also allows the registration /reregistration of URL's for the Web Push Service.

The following functions can be used within this API Wrapper:

- Sending MT messages using the Web Service
- Receive Delivery Receipts (DLR) using the Web Service
- Receive MO Messages using the Web Service
- Message & Web Address Status (URL/s status used in the Push Service) using the Web Service
- Web Services Registering & Unregistering (URL/s used in the Push Service) using the Web Service

The next section will show how to download and use this wrapper for C#.

The core ***SmsClient*** has the following fields/Properties and Methods:



API DOWNLOAD

Download the SMS API Wrapper (C# or Visual Basic)

1. Create a new C# or Visual Basic project. **NOTE:** the target must be .NET 4.5.2
2. Right click on your project and click “Manage NuGet Packages…”
3. In the “Manage NuGet Packages” dialog, search the “Online” section for **M:Science**.
4. Install the “**M:Science Web Services Client API**” package
5. Use the client library from your code.

Download the SMS API Wrapper (PHP)

1. Create a new PHP project
2. Download “SmsClient.php” from https://github.com/MScience/Dragonfly-php
3. Add “SmsClient.php” to the project.
4. Require “SmsClient.php” within your php file.

Download the SMS API Wrapper (Java)

1. Create a new Java project using Eclipse.
2. Download “SmsClient.Sms.jar” from https://github.com/MScience/Dragonfly-Java
3. Within Eclipse add this as a referenced library for your project.

How to send a single synchronous message?

The below code fragment shows the C# call to send a message:

```
var client = new MScience.Sms.SmsClient
{
      AccountId = "YourAccountId",
      Password = "YourPassword"
};

var sendResult = client.Send(            "+44123456789", //destination number
                                         "myCompany",   // source
                                "Hello world", //message
                                         0,          //source Id
                                    true);       //delivery receipt
```

The below code fragment shows the Visual Basic call to send a message:

```
Imports MScience.Sms

Module Module1

    Sub Main()
        Dim client As New SmsClient()
        client.AccountId = "YourAccountId"
        client.Password = "YourPassword"

        Dim sendResult As SendResult = client.Send(
                              "+44123456789", 'destination number
                             "", 'source
                             "Hello world", 'message
                             0, 'source id
                             True) 'delivery receipt

    End Sub


End Module
```

The below code fragment shows the PHP call to send a message:

```
<?php
require 'SmsClient.php';
$client = new MScience\SmsClient('YourAccountId','YourPassword');
$result2 = $client->send(array(new MScience\SmsMessage
                                        ('myCompany',   //source
            '+44123456789',//destination number
            0,          //source Id
            'Hello world'  //message
            ,true)));     //delivery receipt

?>
```

The below code fragment shows the Java call to send a message:

```
import com.mscience.*;

public class SendMessage{
          public static void main(String[] args) throws Exception, Exception {

   SmsClient client = new SmsClient("YourAccountId","YourPassword");
   SendResult result = client.send("+44123456789",//destination number
                                "", //source
                               "Hello world", //message
                                0, //source Id
                                true); //delivery receipt
        }
}
```

The '***source***' is the number or text shown as the source of the message when viewed on the destination handset, and can be

either

A string of alphanumeric characters (maximum 11) i.e. JonnyCabs, MikeMech. MT messages sent with this type of source cannot be replied to be the recipient.

or

A valid virtual number that is supplied by DragonFly; no other number can be used other than ones assigned to your account to prevent spoofing. This can contain the full country code as well and not limited by it. I.e. 447123456789, 417646464646. MT messages sent with this type of source can be replied to by the recipient; the replies (MO messages) will be routed according to how your Dragonfly account is set up on our servers.

Note: If the source is left blank, then the source used for the message will be set to the default source configured on your account. This default can be set/changed upon request.

A virtual number can be provided on trial for 2 weeks. Please see the current site the most up to date prices for renting numbers after your trial period is up.

How to send a single asynchronous message?

Sending an asynchronous message C# example code

```
var client = new MScience.Sms.SmsClient
        {
           AccountId = "YourAccountId",
           Password = "YourPassword"
        };

client.SendAsync("+447123456789",    //destination
            "AsyncTest",          //  source
            "Async Simple Test", // mesage
             22,     // delivery receipt
 false).ContinueWith(result => WriteSendResult(result.Result, client)).Wait();


private static void WriteSendResult(SendResult sendResult, ISmsClient client)
      {
         Console.WriteLine(sendResult.Code);
         if (sendResult.HasError)
         {
            Console.WriteLine(sendResult.ErrorMessage);
```

```
        }
        else
        {
            Console.WriteLine("Message Id {0}", sendResult.MessageId);
            Console.WriteLine("Balance {0}", sendResult.MessageBalance);
            Console.WriteLine("Pending : {0}", sendResult.PendingMessages);
            Console.WriteLine("Surcharge : {0}",  sendResult.SurchargeBalance);
            var statusResult = client.GetMessageStatus(new[] { sendResult.MessageId });
            Console.WriteLine("Message Status Code = {0}, Status = {1}",
statusResult.First().Code, statusResult.First().Status);
        }

    }
```

Sending an asynchronous message Visual Basic example code

```
Imports MScience.Sms

Module Module1

  Sub Main()
    Dim client As New SmsClient()

    client.AccountId = "YourAccountId"
    client.Password = "YourPassword"

    client.SendAsync("+44123456789", 'destination number
                                     "", 'source
                                     "Hello world", 'message
                                     0, 'source id
                                     True).ContinueWith(Sub(result)
WriteSendResult(result.Result, client)).Wait() 'delivery receipt

  End Sub

  Sub WriteSendResult(ByRef SendResult As SendResult, ByRef client As ISmsClient)

    Console.WriteLine(SendResult.Code)
    If (SendResult.HasError) Then
      Console.WriteLine(SendResult.ErrorMessage)
    Else
      Console.WriteLine("Message Id {0}", SendResult.MessageId)
      Console.WriteLine("Balance {0}", SendResult.MessageBalance)
      Console.WriteLine("Pending : {0}", SendResult.PendingMessages)
      Console.WriteLine("Surcharge : {0}", SendResult.SurchargeBalance)

      Dim status As StatusResult()
      status = client.GetMessageStatus({SendResult.MessageId})

      Console.WriteLine("Message Status Code = {0}, Status = {1}",
                status.First().Code, status.First().Status)
    End If
  End Sub
```

End Module

Quick Start. To begin using the SMS Web Service, the first step is to set up an online Dragonfly SMS account. Go to http://www.dragonflysms.com/sign-up/ and complete the registration form to register an account and receive 100 free message credits

Multiple web service servers. The web service is available on two addresses. These are smswebservice1.m-science.com and smswebservice2.m-science.com. This enables the possibility of failing over should one of the servers be unavailable. The SMS Wrapper API handles this within its method calls so you don't have to consider it manually.

API Wrapper. The table summarises the core functionality of the API wrappers below:

Core Functionality	**Core Functionality Description**
Sending Messages	Sending single and bulk synchronous/asynchronous MT SMS message/s
Delivery Receipts (DLR)	Retrieve delivery receipts and asynchronous delivery receipts for previously sent MT messages
Receiving Messages	Retrieves MO messages and asynchronous MO messages
Message and Web Address status	Get the status of each message including asynchronously and get the status of an inbound address including asynchronously
Web Services Registration & Unregistering, Clear Down	Register/unregister web service addresses for inbound notifications including asynchronously. Clears all registered web service addresses from receiving inbound notifications including asynchronously.

CODE WRAPPER API EXAMPLES

SENDING MT MESSAGES

Sending a single synchronous/asynchronous message or send bulk synchronous/asynchronous messages using the method ***Send.***

The messages will be filtered for unknown characters The source address will appear on the end user's mobile handset, and will be used for any reply.

If a delivery receipt is requested the caller must pass a unique message identifier in the SourceId for each message. This will be sent back with the receipt to allow it to be correlated to the original message. See the Delivery Receipt Section for further details.

Concatenated messages

If a message is sent that is greater than 160 characters, then during transmission to the mobile networks the message is split into 160 character messages. The messages are then re-assembled on the destination handset into one contiguous message. This is important to note as messages sent that are greater than 160 characters will consume more than one message credit from your account. Maximum length of messages supported by our system is 918 characters.

Sending a single message

Use the ***Send*** method with the 5 parameters within the call.

If you leave the source empty, then the source address will be picked up from your main account settings. Only add a source if you wish to override this.

The return from Send is an instance of the ***SendResult*** class, which gives you a Message ID and also information about your account including your new message balance.

To find out if the message was accepted and is being processed, then you need to call the ***GetMessageStatus*** method on the client class – passing in the Message ID - which returns a ***StatusResult*** class.

Sending a synchronous message C# example code

```
 var client = new MScience.Sms.SmsClient
        {
           AccountId = "YourAccountId",
           Password = "YourPassword"
        };
        var sendResult = client.Send("44123456789",   //destination
                              "MyCompany",
//source
```

```
                        "Hello world",              //message
                        0,                          //sourceId
                        true);                      //deliveryReceipt

        Console.WriteLine(sendResult.Code);
        if (sendResult.HasError)
        {
            Console.WriteLine(sendResult.ErrorMessage);
        }
        else
        {
            Console.WriteLine("Message Id {0}", sendResult.MessageId);
            Console.WriteLine("Balance {0}",
sendResult.MessageBalance);
            Console.WriteLine("Pending : {0}",
sendResult.PendingMessages);
            Console.WriteLine("Surcharge : {0}",
sendResult.SurchargeBalance);
            var status = client.GetMessageStatus(new[] {
sendResult.MessageId });
            Console.WriteLine("Message Status Code = {0}, Status = {1}",
                    status.First().Code, status.First().Status);
        }
```

Sending a synchronous message Visual Basic example code

```
Imports MScience.Sms

Module Module1

    Sub Main()
        Dim client As New SmsClient()
        client.AccountId = "YourAccountId"
        client.Password = "YourPassword"

        Dim sendResult As SendResult = client.Send("+44123456789",
'destination number
                                    "", 'source
                                    "Hello world", 'message
                                    0, 'source id
                                    True) 'delivery receipt

        Console.WriteLine(sendResult.Code)

        If (sendResult.HasError) Then
            Console.WriteLine(sendResult.ErrorMessage)
        Else
            Console.WriteLine("Message Id {0}", sendResult.MessageId)
            Console.WriteLine("Balance {0}", sendResult.MessageBalance)
```

```
            Console.WriteLine("Pending : {0}",
sendResult.PendingMessages)
            Console.WriteLine("Surcharge : {0}",
sendResult.SurchargeBalance)

            Dim status As StatusResult()
            status = client.GetMessageStatus({sendResult.MessageId})

            Console.WriteLine("Message Status Code = {0}, Status = {1}",
                    status.First().Code, status.First().Status)
        End If

    End Sub

End Module
```

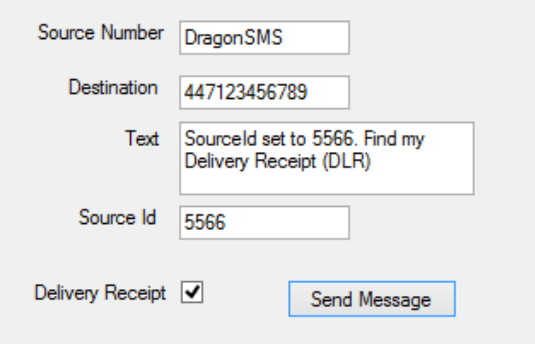
Sending a synchronous message PHP example code

```
<?php
require 'SmsClient.php';
$client = new MScience\SmsClient('YourAccountId','YourPassword');
$result2 = $client->send(array(new MScience\SmsMessage
                        ('myCompany',       //source
                         '+44123456789',    //destination number
                         0,                   //source Id
                         'Hello world'       //message
                         ,true)));           //delivery receipt

echo "Code: ".$result2[0]->Code. "<br>";

if (!empty($result2[0]->ErrorMessage))
{
    echo "Error Message: " .$result2[0]->ErrorMessage. "<br>";
}
else
{
    echo "Message Id: " .$result2[0]->MessageId. "<br>";
    echo "Balance: " .$result2[0]->MessageBalance. "<br>";
    echo "Pending: " .$result2[0]->PendingMessages. "<br>";
    echo "Surcharge: " .$result2[0]->SurchargeBalance. "<br>";
    $status = $client->getMessageStatus(array($result2[0]->MessageId));
    echo "Message Status Code = " .$status[0]->Code. ", SubCode =
" .$status[0]->SubCode. "<br>";
}
?>
```

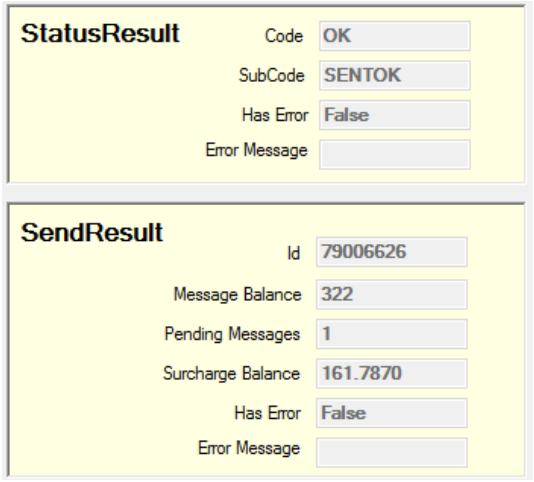
Sending a synchronous message Java example code

```
import com.mscience.*;

public class SendMessage {
	public static void main(String[] args) throws Exception,
Exception {

	SmsClient client = new
SmsClient("YourAccountId","YourPassword");
   SendResult result = client.send("+44123456789",//destination number
                                   "", //source
                                  "Hello world", //message
                                   0, //source Id
                                   true); //delivery receipt

                        if (result.HasError()){
                           System.out.println("");
                           System.out.printf("Code: %s, Error: %s",
                                    result.getCode(), result.getErrorMessage());
                        }
                        else{
                           System.out.printf("Code: %s, Id: %d, Balance: %d",
                                    result.getCode(),
                                    result.getMessageId(),
                                    result.getMessageBalance());
                           System.out.printf("Pending: %d, Surcharge: %.6f",
                                    result.getPendingMessages(),
                                    result.getSurchargeBalance());

                           StatusResult[] status = client.getMessageStatus(new
int[] {result.getMessageId()});

                           System.out.printf("Message Status Code: %s,
Status: %s",
                                       status[0].getCode(),
                                       status[0].getStatus());

                           System.out.println("");
                        }
	}
}
```

Send a message



Result from the class StatusResult and SendResult



The ***StatusResult*** class has the following fields

Public fields	Description
Code	Return Status Code
SubCode	The sub code of the result returned from the web service
HasError	True if an error was returned from the Web Service
ErrorMessage	Error message returned from the Web Service

The ***SendResult*** class has the following fields

Public fields	Description
Messageid	The Id of the Message or Delivery Receipt
MessageBalance	Remaining message balance
PendingMessages	Outbound messages waiting to be sent
SurchargeBalance	Remaining surcharge balance
ErrorMessage	Error message returned from the Web Service
HasError	True if an error was returned from the Web Service

NB. The Code field has the code and subcode of the message for example OK-SENTOK for the above example.

Sending an asynchronous message C# example code

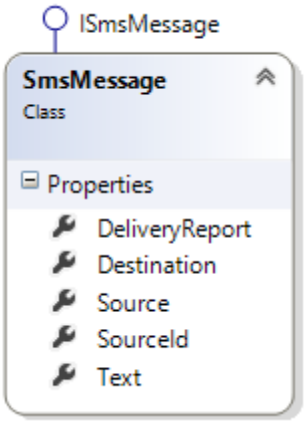
```
var client = new MScience.Sms.SmsClient
        {
           AccountId = "YourAccountId",
           Password = "YourPassword"
        };

 client.SendAsync("+447123456789",    //destination
              "AsyncTest",          //  source
              "Async Simple Test", // mesage
               22,     // delivery receipt
  false).ContinueWith(result => WriteSendResult(result.Result, client)).Wait();


 private static void WriteSendResult(SendResult sendResult, ISmsClient client)
      {
         Console.WriteLine(sendResult.Code);
         if (sendResult.HasError)
         {
            Console.WriteLine(sendResult.ErrorMessage);
         }
         else
         {
            Console.WriteLine("Message Id {0}", sendResult.MessageId);
            Console.WriteLine("Balance {0}", sendResult.MessageBalance);
            Console.WriteLine("Pending : {0}", sendResult.PendingMessages);
            Console.WriteLine("Surcharge : {0}",  sendResult.SurchargeBalance);
            var statusResult = client.GetMessageStatus(new[] { sendResult.MessageId });
            Console.WriteLine("Message Status Code = {0}, Status = {1}",
 statusResult.First().Code, statusResult.First().Status);
         }

      }
```

Sending an asynchronous message Visual Basic example code

```
Imports MScience.Sms

Module Module1

    Sub Main()
        Dim client As New SmsClient()

        client.AccountId = "YourAccountId"
        client.Password = "YourPassword"

        client.SendAsync("+44123456789", 'destination number
                                         "", 'source
                                         "Hello world", 'message
                                         0, 'source id
                                         True).ContinueWith(Sub(result)
WriteSendResult(result.Result, client)).Wait() 'delivery receipt

    End Sub

    Sub WriteSendResult(ByRef SendResult As SendResult, ByRef client As ISmsClient)

        Console.WriteLine(SendResult.Code)
        If (SendResult.HasError) Then
            Console.WriteLine(SendResult.ErrorMessage)
        Else
            Console.WriteLine("Message Id {0}", SendResult.MessageId)
            Console.WriteLine("Balance {0}", SendResult.MessageBalance)
            Console.WriteLine("Pending : {0}", SendResult.PendingMessages)
            Console.WriteLine("Surcharge : {0}", SendResult.SurchargeBalance)

            Dim status As StatusResult()
            status = client.GetMessageStatus({SendResult.MessageId})

            Console.WriteLine("Message Status Code = {0}, Status = {1}",
                        status.First().Code, status.First().Status)
        End If
    End Sub

End Module
```

Sending bulk messages

Use the ***Send*** method with the 1 parameter as an array of ***SmsMessage*** which is a simple object**.** This can be one of many and as long as it is an array the service will handle it.



Bulk sending messages C# example code. This sample shows 3 messages being sent, which is handled by a loop to simulate 3 messages.

```
var client = new MScience.Sms.SmsClient
{
        AccountId = "YourAccountId",
        Password = "YourPassword"
 };

var loopCount = 3;

var messages = new List<SmsMessage>();

for (int i = 0; i < loopCount; i++)
{
        messages.Add(
            new SmsMessage
            {
                Source = "MyCompany",
                Text = String.Format("Test message {0}", i),
                DeliveryReport = true,
                SourceId = 98+i,
                Destination = "+447123456789"
             });
 }

 var sendResults = client.Send(messages.ToArray());

 if (sendResults.Length > 0)
 {
```

```
            Console.WriteLine("Send Results: ");

                    foreach (var sendResult in sendResults)
                    {
                     Console.WriteLine("Message Id {0}", sendResult.MessageId);
                     Console.WriteLine("Balance {0}",
 sendResult.MessageBalance);
                     Console.WriteLine("Pending : {0}",
 sendResult.PendingMessages);
                     Console.WriteLine("Surcharge : {0}",
 sendResult.SurchargeBalance);
                     var status = client.GetMessageStatus(new[] {
 sendResult.MessageId });
                     Console.WriteLine("Message Status Code = {0}, Status = {1}",
                      status.First().Code, status.First().Status);
            }
 }
```

Bulk sending messages Visual Basic example code.

```
Imports MScience.Sms

Module Module1

   Sub Main()
      Dim client As New SmsClient()

      client.AccountId = "YourAccountId"
      client.Password = "YourPasword"

      Dim loopCount As Integer = 3

      Dim messages As New List(Of SmsMessage)

      For index = 1 To loopCount

         Dim message As New SmsMessage
         message.Source = "My Company"
         message.Text = "Test Message " + index.ToString()
         message.DeliveryReport = True
         message.SourceId = 98 + index
         message.Destination = "+44123456789"

         messages.Add(message)

      Next

      Dim sendResults As SendResult()
      sendResults = client.Send(messages.ToArray())
```

```
    If sendResults.Length > 0 Then

      Console.WriteLine("Send Results: ")

      For Each sendResult As SendResult In sendResults
        Console.WriteLine("Message Id {0}", sendResult.MessageId)
        Console.WriteLine("Balance {0}", sendResult.MessageBalance)
        Console.WriteLine("Pending : {0}", sendResult.PendingMessages)
        Console.WriteLine("Surcharge : {0}", sendResult.SurchargeBalance)

        Dim status As StatusResult()
        status = client.GetMessageStatus({sendResult.MessageId})

        Console.WriteLine("Message Status Code = {0}, Status = {1}",
                status.First().Code, status.First().Status)
      Next

    End If
  End Sub

End Module
```

Bulk sending messages PHP example code.

```
<?php
require 'SmsClient.php';
$client = new MScience\SmsClient('YourAccountId','YourPassword');

$loopCount = 3;

$messages = array();

for ($x = 1; $x <= $loopCount; $x++)
{
   $messages[] = new MScience\SmsMessage
                        ('MyCompany',   //source
                         '+44123456789',//destination number
                         98 + $x,           //source Id
                         "Text message $x "  //message
                         ,true);   //delivery receipt
}

$sendResults = $client->send($messages);

if (count($sendResults) > 0)
{
    echo "Send Results: “;

   foreach ($sendResults as $sendResult)
```

```
  {
    echo "Message Id: " .$sendResult->MessageId. "<br>";
    echo "Balance: " .$sendResult->MessageBalance. "<br>";
    echo "Pending: " .$sendResult->PendingMessages. "<br>";
    echo "Surcharge: " .$sendResult->SurchargeBalance. "<br>";
    $status = $client->getMessageStatus(array($sendResult->MessageId));
    echo "Message Status Code = " .$status[0]->Code. ", SubCode =
" .$status[0]->SubCode. "<br>";
  }
}

?>
```

Bulk sending messages Java example code.

```
import java.util.ArrayList;
import com.mscience.*;

public class SendBulkMessages{
    public static void main(String[] args) throws Exception, Exception {

    SmsClient client = new SmsClient("YourAccountId","YourPassword");

    int endValue = 3;

    ArrayList<SmsMessage> messages = new ArrayList<SmsMessage>();

    for (int loopVal =0; loopVal < endValue; loopVal++){

            messages.add(new SmsMessage
                        ("+44123456789", //destination number
                                    "", //source
                                    "Test Message" + loopVal,
//message
                                    0, //source Id
                                    true)); //          delivery
receipt);
    }

    SmsMessage[] smsMessages = new SmsMessage[messages.size()];

    for(int i =0; i <messages.size(); i++)
    {
            smsMessages[i] = messages.get(i);
    }

    SendResult[] results = client.send(smsMessages);

    if (results.length > 0){

       System.out.println("send Results: ");
```

```
        for (int loopNum =0; loopNum < results.length; loopNum++){

            System.out.println("");
                System.out.printf("Code: %s, Id: %d, Balance: %d",
                results[loopNum].getCode(),
                results[loopNum].getMessageId(),
                results[loopNum].getMessageBalance());
                System.out.printf("Pending: %d, Surcharge: %.6f",
                                results[loopNum].getPendingMessages(),
              results[loopNum].getSurchargeBalance());

                StatusResult[] status = client.getMessageStatus(new int[]
{results[loopNum].getMessageId()});

                System.out.printf("Message Status Code: %s, Status: %s",
                            status[0].getCode(),
                            status[0].getStatus());

                System.out.println("");
        }
    }
  }
}
```

DELIVERY RECEIPT (DLR)

Use the ***GetDeliveryReceipts()*** method with no parameters, which retrieves up to 5 DLRs from the Web Service for your account..

This returns an instance of the ***InboundMessageResult*** class, with the ***DeliveryReceipt*** property set to true. The delivery receipt is contained within the ***Text*** property. The **InboundMessageResult** class is shared with MO messages (see next section) and the properties mean slightly different things depending on the setting of the **DeliveryReceipt** property.

To match a delivery receipt to a previously sent MT message, compare the Source ID within the class to the Source IDs you previously passed when sending the messages via the API.

Retrieving the delivery receipts C# example code

```
var client = new MScience.Sms.SmsClient
        {
           AccountId = "YourAccountId",
           Password = "YourPassword"
        };

        var deliveryReceipts = client.GetDeliveryReceipts();

        if (deliveryReceipts.Length > 0)
        {
           Console.WriteLine("Delivery Receipts: ");

           foreach (var deliveryReceipt in deliveryReceipts)
           {
              Console.WriteLine("Code            " + deliveryReceipt.Code);
              Console.WriteLine("Delivery Receipt  " +
deliveryReceipt.DeliveryReceipt);
              Console.WriteLine("Destination      " +
deliveryReceipt.Destination);
              Console.WriteLine("Error Message     " +
deliveryReceipt.ErrorMessage);
              Console.WriteLine("Has Error        " + deliveryReceipt.HasError);
              Console.WriteLine("Id              " + deliveryReceipt.Id);
              Console.WriteLine("Received        " + deliveryReceipt.Received);
              Console.WriteLine("Source          " + deliveryReceipt.Source);
              Console.WriteLine("SourceId        " + deliveryReceipt.SourceId);
              Console.WriteLine("Text            " + deliveryReceipt.Text);
        }
   }
```

Retrieving the delivery receipts Visual Basic example code

```
Imports MScience.Sms

Module Module1

   Sub Main()
      Dim client As New SmsClient()

      client.AccountId = "YourAccountId"
      client.Password = "YourPassword"
```

```
        Dim sendResult As SendResult = client.Send("+44123456789",
'destination number
                                    "", 'source
                                    "Hello world", 'message
                                    0, 'source id
                                    True) 'delivery receipt

        Dim deliveryReceipts As InboundMessageResult()
        deliveryReceipts = client.GetDeliveryReceipts()

        If (deliveryReceipts.Length > 0) Then

            Console.WriteLine("Delivery Receipts:")

            For Each deliveryReceipt As InboundMessageResult In
deliveryReceipts

                Console.WriteLine("Code           " + deliveryReceipt.Code)
                Console.WriteLine("Delivery Receipt  " +
deliveryReceipt.DeliveryReceipt.ToString())
                Console.WriteLine("Destination      " + deliveryReceipt.Destination)
                Console.WriteLine("Error Message     " +
deliveryReceipt.ErrorMessage)
                Console.WriteLine("Has Error        " +
deliveryReceipt.HasError.ToString())
                Console.WriteLine("Id             " + deliveryReceipt.Id.ToString())
                Console.WriteLine("Received        " +
deliveryReceipt.Received.ToString())
                Console.WriteLine("Source          " + deliveryReceipt.Source)
                Console.WriteLine("SourceId        " +
deliveryReceipt.SourceId.ToString())
                Console.WriteLine("Text           " + deliveryReceipt.Text)

            Next

        End If
    End Sub

End Module
```

Retrieving the delivery receipts PHP example code

```
<?php
require 'SmsClient.php';
$client = new MScience\SmsClient('YourAccountId','YourPassword');

$deliveryReceipts = $client->getDeliveryReceipts();

if (count($deliveryReceipts) > 0)
```

```
{
   echo "Delivery Receipts: ";

   foreach ($deliveryReceipts as $deliveryReceipt)
   {
      echo "Code: " .$deliveryReceipt->Code. "<br>";
      echo "Delivery Receipt: " .$deliveryReceipt->DeliveryReceipt. "<br>";
      echo "Destination: " .$deliveryReceipt->Destination. "<br>";
      echo "Error Message: " .$deliveryReceipt->ErrorMessage. "<br>";
      echo "Has Error: " .$deliveryReceipt->HasError. "<br>";
      echo "Id: " .$deliveryReceipt->Id. "<br>";
      echo "Received: " .$deliveryReceipt->Received. "<br>";
      echo "Source: " .$deliveryReceipt->Source. "<br>";
      echo "SourceId: " .$deliveryReceipt->SourceId. "<br>";
      echo "Text: " .$deliveryReceipt->Text. "<br>";

   }
}

?>
```

Retrieving the delivery receipts Java example code

```
import com.mscience.*;

public class DeliveryReceipts{
        public static void main(String[] args) throws Exception, Exception
{

                SmsClient client = new
SmsClient("YourAccountId","YourPassword");

                InboundMessageResult[] deliveryReceipts =
client.getDeliveryReceipts();

                if (deliveryReceipts.length > 0){
                        System.out.println("Delivery Receipts: ");

                        for(InboundMessageResult message :
deliveryReceipts){
                           System.out.printf("Code: %s, Id: %d, Source: %s,
Destination: %s",
                                        message.getCode(),
                                        message.getId(),
                                        message.getSource(),
                                        message.getDestination());
```

```
                                System.out.printf("Received: %s, SourceId: %d,
IsReceipt: %s, text: %s",
                                        message.getReceived(),
                                        message.getSourceId(),
                                        message.getDeliveryReceipt(),
                                        message.getText());
                                System.out.println("");
                            }
                        }

                }
}
```

The Text property contains the details of the sent message including the delivery status. The delivery status follows after the keyword ***stat: - look for this keyword and the status will be one of the following table***

Table: Delivery receipts types

Status	Description
DELIVRD	Message was delivered to cell phone.
SENT	Message was sent to the operator but yet to get any DLR
ACKED	Message was sent to the operator and acknowledged by the operator
FAILED	After attempting for a period of time, the carrier has failed to deliver the message to the destination handset; might be caused by an invalid/malformed source address, Mobile handset turned off or out of reach. Ensure the number is correct, also the handset within range and is switched on.
ACCEPTED	Message was accepted by the operator for delivery
ENQUEUE	Message was sent to the operator and forwarded to the cell phone but yet to reply with Delivery Acknowledgement.
ENROUTE	Message is being routed for delivery
UNKNOWN	The message has been confirmed as undelivered but no detailed information related to the failure is known.
	An unknown error might have occurred on the operator SMS server.
	This can also be generated as a result of PORTABILITY error
UNDELIV	Attempt to deliver message failed.
	Handset might be out of network coverage when message delivery was attempted.
	Handset might be switched off.
	Handset might be roaming out of carrier's supported network.

	The message might have failed due to suspicion of SPAM on the operator network e.g. sender ID relating to financial institutions or government agencies.
	Random unsolicited messages blocked by the operator
	Political campaign not permitted by the operator
	Same message sent to the same phone number repeatedly over a short time
	Absent phone number. The phone number might have been recycled after a period of no activity
	Low or no network coverage
	Battery too low to process incoming message
EXPIRED	The SMSC was unable to deliver the message in a specified amount of time. This is called Message Validity and it is usually 48 hours.
	In some instances, (especially when the operator is under traffic pressure), the operator might expire some messages instantly once it cannot be delivered as at first attempt.
	It can also mean that the cell phone was switched off as at the time of delivery.
REJECTD	The message was rejected.
	The provider could have blocked phone numbers in this range.
	Message flagged as SPAM (Mass unsolicited messages)
	Message flagged as SCAM (Messages with potential intent to defraud, defame or threatens social security)
	Faulty or ongoing upgrade on the base station covering the range of the phone number
DELETED	Message was deleted
UNSUPPORTED	Delivery report is not supported by the route or the operator

Some of the most common status codes:

> DELIVRD, ACKED, UNDEL, FAILED, UNKNOWN, REJECTD, EXPIRED

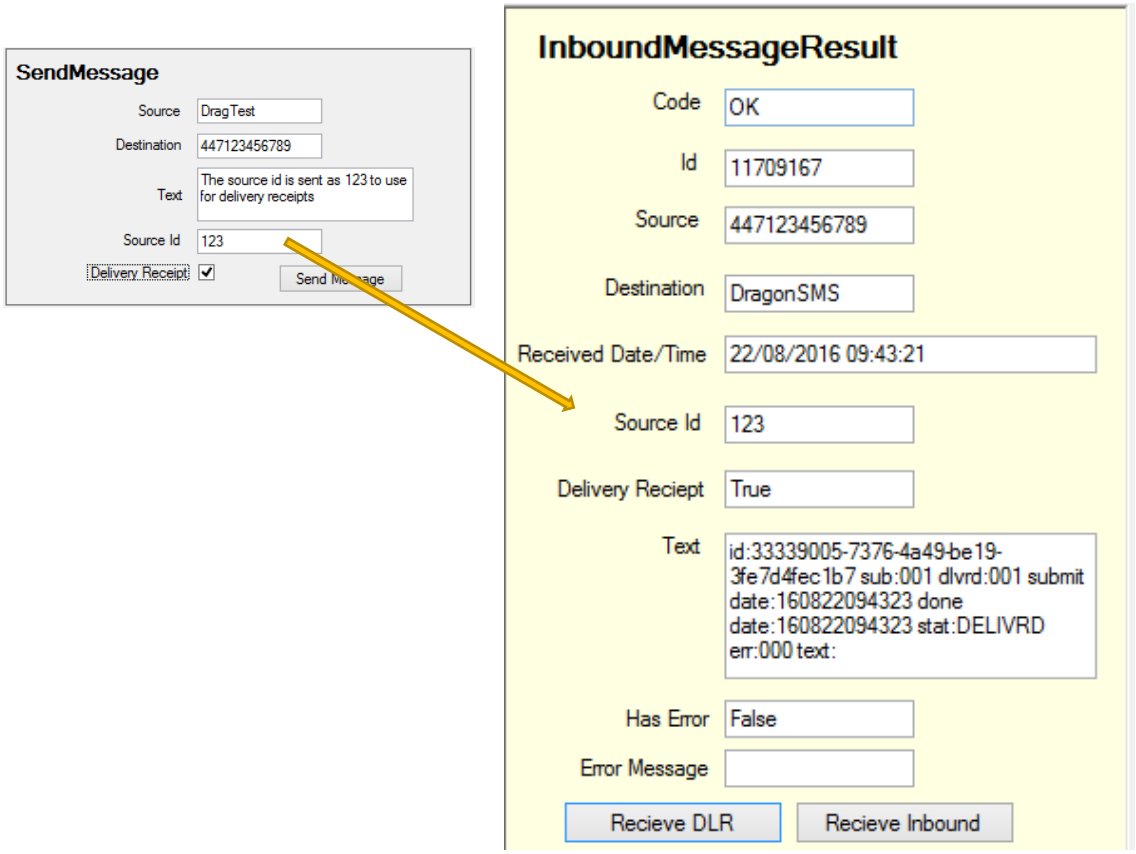
Here is a typical string of the property ***text***

> *Id:33339005-7376-4a49-be19-3fe7d4fec1b7 sub:001 dlvrd:001 submit date:160822094323 done date:160822094323* ***stat:DELIVRD*** *err:000 text:*

Below depicts the yellow arrow showing both the send Source Id and the inbound message delivery receipt Source Id.

Send a message

Result from the class InboundMessageResult



class ***InboundMessageResult***

Field	Description
Code	Return Status Code
Id	The Id of the Delivery Receipt
Source	The Source of the Delivery Receipt
Destination	The Destination of the Delivery Receipt
Received	The Received Date/Time of the Delivery Receipt
SourceId	The Source Id of the Delivery Receipt. Can be used to correlate between inbound and outbound messages
DeliveryReceipt	Is this result a delivery receipt

Text	The text of the Delivery Receipt. The delivery status is within this attribute after the keyword *stat:*
HasError	True if an error was returned from the Web Service
ErrorMessage	Error message returned from the Web Service

The ***Text*** attribute can have the following text, highlighted in blue and this text can vary owing to the different available routes that the MScience/Dragonfly traffic takes. The Id: field can be of differing formats and so it best to parse on the key fields i.e. id:, sub:,dlvrd:, submit date:, done date:, stat:, err:, text:

The delivery status is stated post the 'stat:' text shown here in bold.

> OK-11628655,+447123456789,dispSource,16/08/2016 11:01:23,1237,1,16/08/2016 11:01:23,102,id:3471301395 sub:001 dlvrd:001 submit date:1608161001 done date:1608161001 stat:**DELIVRD** err:004 text:,,

> OK-11628674,+447123456789,dispSource,16/08/2016 11:03:03,1240,1,16/08/2016 11:03:03,102,id:3651338785 sub:001 dlvrd:001 submit date:1608161003 done date:1608161003 stat:**ACKED** err:003 text:,,

RECEIVING MO MESSAGES

You need a virtual number assigned to your account and the calls to retrieve these messages.

To retrieve MO messages, use the method ***GetInboundMessages*** with no parameters, which retrieves up to 5 DLRs from the Web Service for your account.

class ***InboundMessageResult***

Field	**Description**
Code	Return Status Code
Id	The Id of the Message
Source	The Source of the Message
Destination	The Destination of the Message
Received	The Received Date/Time of the Message
SourceId	The Source Id of the Message
DeliveryReceipt	Is this result a delivery receipt or a message
Text	The text of the Message

HasError	True if an error was returned from the Web Service
ErrorMessage	Error message returned from the Web Service

Retrieving the MO message/s C# example code. This returns an array of ***InboundMessages[]***

Use the ***GetInboundMessages ()*** method with no parameters

```
    var client = new MScience.Sms.SmsClient
    {
            AccountId = "YourAccountId",
            Password = "YourPassword"
     };
    var inboundMessages = client.GetInboundMessages();

     if (inboundMessages.Length > 0)
     {
            Console.WriteLine("Inbound Messages: ");

            foreach (var inboundMessage in inboundMessages)
            {
                if (inboundMessage.Destination != "")
                {
                  Console.WriteLine("Code: ", inboundMessage.Code);
                  Console.WriteLine("Id: ", inboundMessage.Id);
                  Console.WriteLine("Delivery Receipt: ",
inboundMessage.DeliveryReceipt);
                  Console.WriteLine("Destination: ", inboundMessage.Destination);
                  Console.WriteLine("Source: ", inboundMessage.Source);
                  Console.WriteLine("Received: ", inboundMessage.Received);
                  Console.WriteLine("Has Error: ", inboundMessage.HasError);
                  Console.WriteLine("SourceId: ", inboundMessage.SourceId);
                  Console.WriteLine("Text: ", inboundMessage.Text);
            }
      }
}
```

Retrieving the MO message/s Visual Basic example code.

```
Imports MScience.Sms

Module Module1

   Sub Main()
```

```
        Dim client As New SmsClient()

        client.AccountId = "YourAccountId"
        client.Password = "YourPassword"

        Dim sendResult As SendResult = client.Send("+44123456789", 'destination number
                                        "", 'source
                                        "Hello world", 'message
                                        0, 'source id
                                        True) 'delivery receipt

        Dim inboundMessages As InboundMessageResult()
        inboundMessages = client.GetInboundMessages()

        If (inboundMessages.Length > 0) Then

            Console.WriteLine("Inbound Messages:")

            For Each inboundMessage As InboundMessageResult In inboundMessages

                If Not inboundMessage.Destination = "" Then

                    Console.WriteLine("Code: " + inboundMessage.Code)
                    Console.WriteLine("Delivery Receipt: " +
inboundMessage.DeliveryReceipt.ToString())
                    Console.WriteLine("Destination: " + inboundMessage.Destination)
                    Console.WriteLine("Error Message: " + inboundMessage.ErrorMessage)
                    Console.WriteLine("Has Error: " + inboundMessage.HasError.ToString())
                    Console.WriteLine("Id: " + inboundMessage.Id.ToString())
                    Console.WriteLine("Received: " + inboundMessage.Received.ToString())
                    Console.WriteLine("Source: " + inboundMessage.Source)
                    Console.WriteLine("SourceId: " + inboundMessage.SourceId.ToString())
                    Console.WriteLine("Text: " + inboundMessage.Text)

                End If

            Next

        End If
    End Sub

End Module
```

Retrieving the MO message/s PHP example code.

```
<?php
require 'SmsClient.php';
$client = new MScience\SmsClient('YourAccountId','YourPassword');

$inboundMessages = $client->getInboundMessages();

if (count($inboundMessages) > 0)
{
```

```
    echo "Inbound Messages: ";

    foreach ($inboundMessages as $inboundMessage)
    {
        if (!empty($inboundMessage->Destination))
        {
            echo "Code: " .$inboundMessage->Code. "<br>";
            echo "Id: " .$inboundMessage->Id. "<br>";
            echo "Delivery Receipt: " .$inboundMessage->DeliveryReceipt. "<br>";
            echo "Destination: " .$inboundMessage->Destination. "<br>";
            echo "Source: " .$inboundMessage->Source. "<br>";
            echo "Received: " .$inboundMessage->Received. "<br>";
            echo "HasError: " .$inboundMessage->HasError. "<br>";
            echo "SourceId: " .$inboundMessage->SourceId. "<br>";
            echo "Text: " .$inboundMessage->Text. "<br>";
        }
    }
}

?>
```

Retrieving the MO message/s Java example code.

```
import com.mscience.*;

public class InboundMessages{
        public static void main(String[] args) throws Exception, Exception {

                SmsClient client = new SmsClient("YourAccountId","YourPassword");

                InboundMessageResult[] inboundMessages =
client.getInboundMessages();

                if (inboundMessages.length > 0){
                        System.out.println("Inbound Messages: ");

                        for(InboundMessageResult message : inboundMessages){
                          System.out.printf("Code: %s, Id: %d, Source: %s,
Destination: %s",
                                message.getCode(),
                                message.getId(),
                                message.getSource(),
                                message.getDestination());
                          System.out.printf("Received: %s, SourceId: %d,
IsReceipt: %s, text: %s",
                                message.getReceived(),
                                message.getSourceId(),
                                message.getDeliveryReceipt(),
                                message.getText());
                          System.out.println("");
```

```
                        }
                    }
                }
}
```

WEB SERVICES REGISTRATION & UNREGISTERING

Allows a client to register for the ability to receive MO messages asynchronous inbound message calls. To accept these, there must be http servers at the addresses specified capable of accepting a POST with the parameters described for ReceiveInboundMessage.

Use the method ***RegisterInbound*** passing the ***address*** list as an array for each registered address.

C# example code.

```
var client = new MScience.Sms.SmsClient
{
        AccountId = "YourAccountId",
        Password = "YourPassword"
};

var addresses = new List<string>();
addresses.Add("http://mydomain/Inbound.com");

var registerResults = client.RegisterInbound(addresses.ToArray());
registerResults.ToList().
        ForEach(res => Console.WriteLine("Register Result {0} : {1}", res.Code,
        res.SubCode));
```

Visual Basic example code.

```
Imports MScience.Sms

Module Module1

   Sub Main()
      Dim client As New SmsClient()

      client.AccountId = "YourAccountId"
      client.Password = "YourPassword"

      Dim addresses As New List(Of String)
      addresses.Add("http://mydomain/Inbound.com")
```

```
        Dim registerResults = client.RegisterInbound(addresses.ToArray())

        For Each registerResult As Result In registerResults
            Console.WriteLine("Register Result {0} : {1}", registerResult.Code,
registerResult.SubCode)
        Next

    End Sub

End Module
```

PHP example code.

```
<?php
require 'SmsClient.php';
$client = new MScience\SmsClient('YourAccountId','YourPassword');

$addresses = array();

$addresses[] = "http://mydomain/Inbound.com";

$registerResults = $client->registerInbound($addresses);

foreach ($registerResults as $registerResult)
{
    echo "Register Result " .$registerResult->Code. " : " .$registerResult->SubCode.
"<br>";
}

?>
```

Java example code.

```
import java.util.ArrayList;
import com.mscience.*;

public class RegisterInbound{
        public static void main(String[] args) throws Exception, Exception {

                SmsClient client = new SmsClient("YourAccountId","YourPassword");

                ArrayList<String> addresses = new ArrayList<String>();
                addresses.add("http://mydomain/Inbound.com");

                String[] addressesArr = new String[addresses.size()];

                for(int i =0; i <addresses.size(); i++)
                {
                        addressesArr[i] = addresses.get(i);
                }

                Result[] registerResults = client.registerInbound(addressesArr);
```

```
                    for(int j =0; j <registerResults.length; j++)
                    {
                            System.out.printf("Register Result %s, : %s",
                                    registerResults[j].getCode(),
                                    registerResults[j].getSubCode());
                    }
            }
}
```

Unregisters asynchronous callback addresses specified in ***RegisterInbound***. The addresses must exactly match the ones specified when the callbacks were initially registered.

Use the method ***RemoveInbound*** passing the ***address*** list as an array for each registered address.

C# example code.

```
var client = new MScience.Sms.SmsClient
{
    AccountId = "YourAccountId",
    Password = "YourPassword"
};

var addresses = new List<string>();
addresses.Add("http://mydomain/Inbound.com");

var removeAddressResults = client.RemoveInbound(addresses.ToArray());
removeAddressResults.ToList()
    .ForEach(remResult => Console.WriteLine("Remove Result {0} : {1}",
    remResult.Code, remResult.SubCode));
```

Visual Basic example code.

```
Imports MScience.Sms

Module Module1

  Sub Main()
    Dim client As New SmsClient()

    client.AccountId = "YourAccountId"
    client.Password = "YourPassword"

    Dim addresses As New List(Of String)
    addresses.Add("http://mydomain/Inbound.com")

    Dim removeAddressResults = client.RemoveInbound(addresses.ToArray())
```

```
        For Each removeResult As Result In removeAddressResults
            Console.WriteLine("Register Result {0} : {1}", removeResult.Code,
removeResult.SubCode)
        Next

    End Sub

End Module
```

PHP example code.

```
<?php
require 'SmsClient.php';
$client = new MScience\SmsClient('YourAccountId','YourPassword');

$addresses = array();

$addresses[] = "http://mydomain/Inbound.com";

$removeAddressResults = $client->removeInbound($addresses);

foreach ($removeAddressResults as $removeAddressResult)
{
    echo "Remove Result " .$removeAddressResult->Code. " : " .$removeAddressResult-
>SubCode. "<br>";
}

?>
```

Java example code.

```
import java.util.ArrayList;
import com.mscience.*;

public class RemoveInbound{
        public static void main(String[] args) throws Exception, Exception {

                SmsClient client = new SmsClient("YourAccountId","YourPassword");

                ArrayList<String> addresses = new ArrayList<String>();
                addresses.add("http://mydomain/Inbound.com");

                String[] addressesArr = new String[addresses.size()];

                for(int i =0; i <addresses.size(); i++)
                {
                        addressesArr[i] = addresses.get(i);
                }

                Result[] removeAddressResults =
client.removeInbound(addressesArr);

                for(int j =0; j <removeAddressResults.length; j++)
```

```
                {
                        System.out.printf("Remove Result %s, : %s",
                                        removeAddressResults[j].getCode(),
                                        removeAddressResults[j].getSubCode());
                }
        }
}
```

WEB ADDRESS STATUS

Use the method ***GetInboundStatus*** passing the ***address*** string for each registered address.

C# example code.

```
var client = new MScience.Sms.SmsClient
{
    AccountId = "YourAccountId",
    Password = "YourPassword"
};

var addresses = new List<string>();
addresses.Add("http://mydomain/Inbound.com");

addresses.ForEach(address =>
{
    var status = client.GetInboundStatus(address);

    Console.WriteLine("Inbound Status {0} {1}:{2}", address, status.Code,
    status.SubCode);
});
```

Visual Basic example code.

```
Imports MScience.Sms

Module Module1

  Sub Main()
    Dim client As New SmsClient()

    client.AccountId = "YourAccountId"
    client.Password = "YourPassword"

    Dim addresses As New List(Of String)
    addresses.Add("http://mydomain/Inbound.com")
```

```
        For Each address As String In addresses

            Dim status = client.GetInboundStatus(address)
            Console.WriteLine("Inbound Status {0} {1}:{2}", address, status.Code,
status.SubCode)

        Next

    End Sub

End Module
```

PHP example code.

```
<?php
require 'SmsClient.php';

$client = new MScience\SmsClient('YourAccountId','YourPassword');

$addresses = array();

$addresses[] = "http://mydomain/Inbound.com";

foreach ($addresses as $address)
{
    $status = $client->getInboundStatus($address);

    echo "Inbound Status " .$address. " " .$status->Code. " : " .$status->SubCode. "<br>";
}

?>
```

Java example code.

```
import java.util.ArrayList;
import com.mscience.*;

public class InboundStatus{
        public static void main(String[] args) throws Exception, Exception {

                SmsClient client = new SmsClient("YourAccountId","YourPassword");

                ArrayList<String> addresses = new ArrayList<String>();
                addresses.add("http://mydomain/Inbound.com");

                for(int i =0; i <addresses.size(); i++)
                {
                        Result status = client.getInboundStatus(addresses.get(i));

                        System.out.printf("Inbound Status %s, %s, : %s",
                                        addresses.get(i),
                                        status.getCode(),
                                        status.getSubCode());
```

```
                }

        }
}
```

URL WEB METHOD

To accept these MO messages from the Web Push Service, there must be http servers at the addresses specified capable of accepting a POST with the parameters described for ReceiveInboundMessage.

Parameters ([] – Denotes an array.)

Encrypt (Boolean) - Specifies that the account and password are encrypted.
AccountID (string) - User ID used to login to the website.
Password (string) - Password used to login to the website.
Address (string[]) - Array of fully qualified web addresses.

E.g.

http://www.mycompany.com/ResponseService.amsx/ReceiveInboundMessage

Return Codes

Note: The actual return code will be a comma separated list of return codes; one for each attempted registration unless there is a global failure. E.g. 'OK-REGISTERED,OK-REGISTERED, FAILED-ALREADYUSED,OK-REG….'

OK-REGISTERED - Registered successfully.

FAILED-BADLOGIN - Unable to log in.

FAILED-NULLPARAM - One or more parameters is NULL.

FAILED-DBERR - Internal database error.

FAILED-NOTLICENSED - Not licensed for this functionality.

FAILED-ALREADYUSED - The specified address is already registered.

DRAGONFLY SMS API REFERENCE

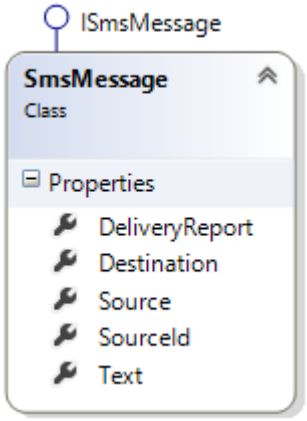
This section provides further details on the types and structures of each class, and associated parameter fields and return types within the package **MScience.Sms** which is a DLL for C#.

This package is located in Visual Studio as under your application tree, and the version number and net version might be different depending on what release you support -

packages\MScience.Sms.1.0.2\lib\net40\MScience.Sms.dll

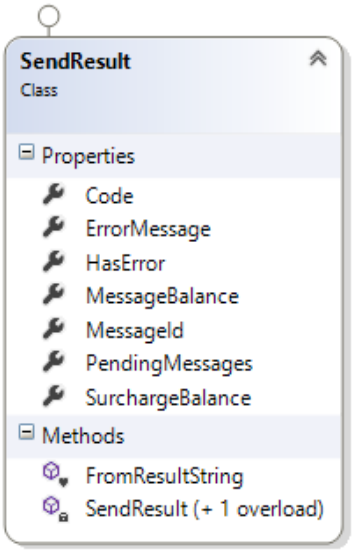
SMSMESSAGE CLASS

Bulk Sending requires the class ***SmsMessage*** array



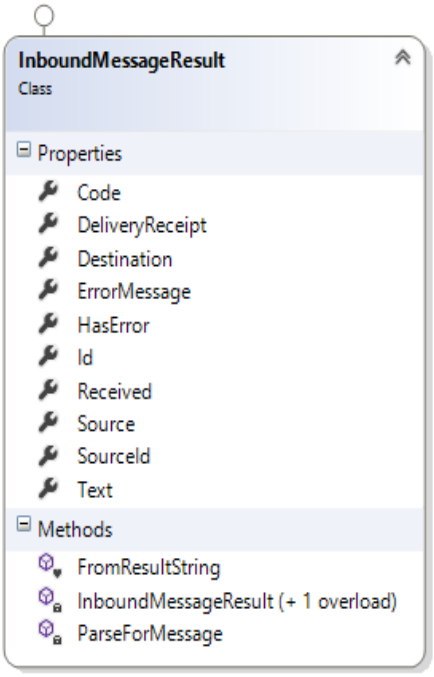
Return Classes

SentResult



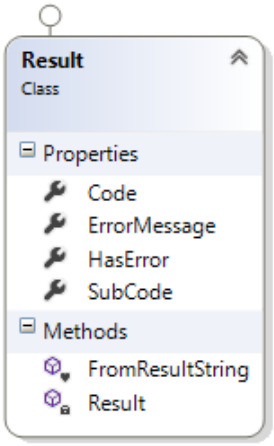
Methods are not exposed and are private

InboundMessageResult



Methods are not exposed and are private

Result



Methods are not exposed and are private

Return Codes

Classes ***SentResult, InboundMessageResult*** and ***Result*** have a return ***Code*** and ***ErrorMessage*** attribute which indicates if the specific web call was successful or failed.

The ***Code*** will be either OK or FAILED and the ***ErrorMessage*** will be the message post pending the return code from the tables 'OK Codes' and 'FAILED Codes' as shown below.

Here is an example for both a OK and FAILED code status with their respective error message:

> ***Code*** = OK-REGISTERED Registration for an inbound web address successful
>
> ***Code*** = FAILED-BADLOGIN Registration for inbound web address has incorrect login details

OK Codes

RETURN CODE	RETURN DESCRIPTION
OK-ACTIVE	Get URL status active
OK-FAILED	Get message status account not web enabled, or Get URL status failed
OK-PENDING	Trying to send message
OK-READY	Get URL status ready
OK-REGISTERED	Registration for an inbound web address successful
OK-RETRY	Get URL status retrying
OK-SENTOK	Message has been sent ok to the SMS Web Service but doesn't dictate that the message was successfully sent to the recipient. (Please see the DLR for the status)

Failed Codes

RETURN CODE	RETURN DESCRIPTION
FAILED	Blocked account – check your account and password, and ensure that the case is correct
FAILED-ALREADYUSED	Register for inbound URL already used
FAILED-BADID	Get message status bad message ID
FAILED-BADLOGIN	Register for inbound bad login details
FAILED-BADPARAMERROR	Send messages Invalid parameter or blocked
FAILED-BLOCKED	The user has failed to login for too many times
FAILED-DBERR	Internal database error
FAILED-NOINTERCONNECTFUND	Send messages insufficient interconnect fund

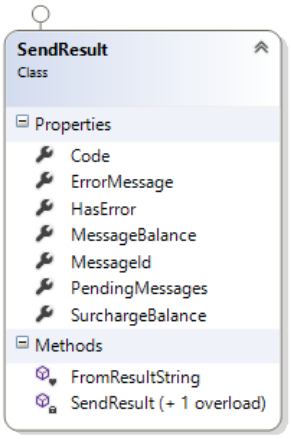
FAILED-NOMESSAGES	Send messages insufficient message balance
FAILED-NOTLICENSED	Account is not web enabled. (Please contact the Dragonfly support desk) or Not licensed for this functionality or Insufficient free registrations Or
FAILED-NULLPARAM	One or more parameters is NULL
FAILED-PARAMERROR	Get message status too many messages specified from host address

SENDING MT MESSAGES

Description	Method Call	Type	Parameter	Return
Synchronously sends a single SMS message	Send	 string string string int bool	 destination source message sourceId deliveryReceipt	SendResult
Asynchronously sends a single message	Send	 string string string int bool	 destination source message sourceId deliveryReceipt	Task<SendResult>
Synchronously sends a number of SMS messages	Send	 SmsMessage[]	 messages	SendResult[]
Asynchronously sends a number of SMS messages	Send	 SmsMessage[]	 messages	Task<SendResult[]>

Class SendResult

Return class ***SendResult*** from Sending Messages



Class SendResult: Public attributes

Public fields	Description	Type
Code	Return Status Code	string
Messageid	Id of the message. Can be used to query for message status	int
MessageBalance	Remaining message balance	int
PendingMessages	Outbound messages waiting to be sent	int
SurchargeBalance	Remaining surcharge balance	decimal
ErrorMessage	Error message returned from the Web Service	string
HasError	True if an error was returned from the Web Service	string

DELIVERY RECEIPTS

Description	Method Call	Parameter	Type	Return
Retrieves delivery receipts from the queue	GetDeliveryReceipts()			InboundMessageResult[]
Asynchronously retrieves	GetDeliveryReceiptsAsync()			Task<InboundMessageResult[

delivery receipts from the queue

class InboundMessageResult

Return class ***InboundMessageResult*** from getting inbound Delivery Receipt Messages

Class InboundMessageResult: Public attributes

Public Field	Description	Type
Code	Return Status Code	string
Id	The Id of the Message or Delivery Receipt	int
Source	The Source of the Message or Delivery Receipt	string
Destination	The Destination of the Message or Delivery Receipt	string
Received	The Received DateTime of the Message or Delivery Receipt	datetime
SourceId	The Source Id of the Message or Delivery Receipt. Can be used to correlate between inbound and outbound messages	Int
DeliveryReceipt	Is this result a delivery receipt or a message	bool
Text	The text of the Message or Delivery Receipt	string
HasError	True if an error was returned from the Web Service	bool

ErrorMessage	Error message returned from the Web Service	string

RECEIVING MO MESSAGES

Description	Method Call	Type	Parameter	Return
Retrieves inbound messages from the queue	GetInboundMessages()	-	-	InboundMessageResult[]
Asynchronously retrieves inbound messages from the queue	GetInboundMessagesAsync()	-	-	Task<InboundMessageResul

Class InboundMessageResult

Return class ***InboundMessageResult*** from getting inbound Messages

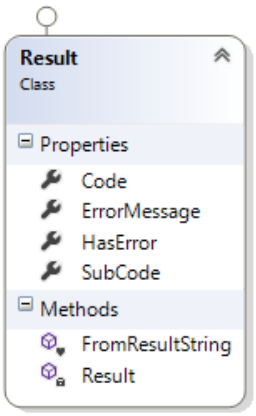
Class InboundMessageResult: Public attributes

Public Field	Description	Type
Code	Return Status Code	string
Id	The Id of the Message or Delivery Receipt	int
Source	The Source of the Message or Delivery Receipt	string
Destination	The Destination of the Message or Delivery Receipt	string
Received	The Received DateTime of the Message or Delivery Receipt	datetime
SourceId	The Source Id of the Message or Delivery Receipt. Can be used to correlate between inbound and outbound messages	Int
DeliveryReceipt	Is this result a delivery receipt or a message	bool
Text	The text of the Message or Delivery Receipt	string
HasError	True if an error was returned from the Web Service	bool
ErrorMessage	Error message returned from the Web Service	string

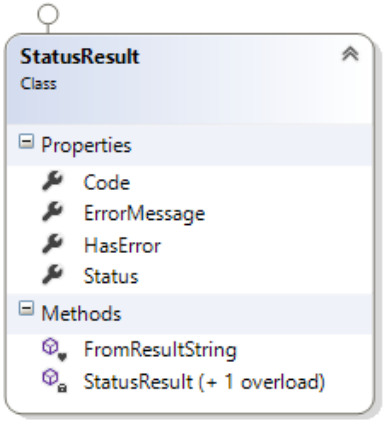
WEB ADDRESS STATUS

Description	Method Call	Type	Parameter	Return
Gets the status of each message in the array	GetMessageStatus	int[]	messages	StatusResult[]
Asynchronously gets the status of each message in the array	GetMessageStatusAsync	int[]	messages	Task<StatusResult[]>
Gets the status of an inbound address	GetInboundStatus	string	address	Result
Asynchronously gets the status of an inbound address	GetInboundStatusAsync	String	address	Task<Result]>

Return class ***Result*** from getting inbound Messages



Return class from ***StatusResult*** from getting inbound Messages



Class Result: Public attributes

Public Field	Description	Type
Code	Return Status Code	string
SubCode	The sub code of the result returned from the web service	string
HasError	True if an error was returned from the Web Service	bool
ErrorMessage	Error message returned from the Web Service	string

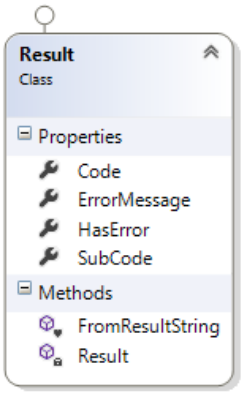
Class Status Result: Public attributes

Public Field	Description	Type
Code	Return Status Code	string
SubCode	The sub code of the result returned from the web service	string
HasError	True if an error was returned from the Web Service	bool
ErrorMessage	Error message returned from the Web Service	string

WEB SERVICES REGISTRATION & UNREGISTERING

Description	Method Call	Type	Parameter	Return
Registers the array of web service addresses for inbound notifications	RegisterInbound	string[]	address	Result[]
Asynchronously registers the array of web service addresses for inbound notifications	RegisterInboundAsync	string[]	addresses	Task<Result[]>
Removes the array of web service addresses from receiving inbound notifications	RemoveInbound	string[]	address	Result[]
Asynchronously removes the array of web service addresses from receiving inbound notifications	RemoveInboundAsync	string[]	addresses	Task<Result[]>
Clears all registered web service addresses from receiving inbound notifications	ClearInbound()	-	-	Result[]
Asynchronously clears all registered web service addresses from receiving inbound notifications	ClearInboundAsync()	-	-	Task<Result[]>

Return class ***Result*** from web services registration & unregistering



Class Result: Public attributes

Public Field	Description	Type
Code	Return Status Code	string
SubCode	The sub code of the result returned from the web service	string
HasError	True if an error was returned from the Web Service	bool
ErrorMessage	Error message returned from the Web Service	string

COMMON HOW TOS

This section shows common scenarios supported by ***SMSClient.***

Send messages?

C# example code.

```
var messages = new List<SmsMessage>();
for (int i = 0; i < loopCount; i++)
{
            messages.Add(
            new SmsMessage
            {
            Source = "447123456789",
                        Text = String.Format("Test message {0}", i),
                        DeliveryReport = true,
                SourceId = 98,
               Destination = "+44713457777"
                });
}
```

Visual Basic example code.

```
Dim messages As New List(Of SmsMessage)

    For index = 1 To loopCount

        Dim message As New SmsMessage
        message.Source = "My Company"
        message.Text = "Test Message " + index.ToString()
        message.DeliveryReport = True
        message.SourceId = 98 + index
        message.Destination = "+44123456789"

        messages.Add(message)

    Next
```

PHP example code.

```
$messages = array();

for ($x = 1; $x <= $loopCount; $x++)
{
   $messages[] = new MScience\SmsMessage
                        ('MyCompany',   //source
                         '+44123456789',//destination number
                         98 + $x,           //source Id
                         "Text message $x "  //message
                         ,true);   //delivery receipt
}
```

Java example code.

```
ArrayList<SmsMessage> messages = new ArrayList<SmsMessage>();

                    for (int loopVal =0; loopVal < endValue; loopVal++){

                            messages.add(new SmsMessage
                                 ("+44123456789", //destination number
                                 "", //source
                                "Test Message" + loopVal, //message
                                 0, //source Id
                                  true); //delivery receipt);
                    }
```

Send asynchronous message?

The function ***WriteSendResult*** will read the results but will be replaced with your own bespoke function.

C# example code.

```
client.SendAsync("+447123456789",

            "+447789456789",

            "Test Simple",

              22,

             false).ContinueWith(result => WriteSendResult(result.Result
client)).Wait();
```

Visual Basic example code.

```
client.SendAsync("+44123456789", 'destination number
                                 "", 'source
                                 "Hello world", 'message
                                 0, 'source id
                                 True).ContinueWith(Sub(result) WriteSendResult(result.Result,
client)).Wait() 'delivery receipt
```

Retrieve delivery receipts?

To capture the DLR's.

C# example code.

```
var deliveryReceipts = client.GetDeliveryReceipts();
if (deliveryReceipts.Length > 0)
{
          Console.WriteLine("Delivery Receipts: ");

            foreach (var deliveryReceipt in deliveryReceipts)
            {
                    Console.WriteLine("Code           " + deliveryReceipt.Code);
                       Console.WriteLine("DeliveryReceipt  " +
deliveryReceipt.DeliveryReceipt);
                       Console.WriteLine("....................

                       Console.WriteLine("Text           " + deliveryReceipt.Text);
            }
}
```

Visual Basic example code.

```
Dim deliveryReceipts As InboundMessageResult()
        deliveryReceipts = client.GetDeliveryReceipts()

        If (deliveryReceipts.Length > 0) Then

            Console.WriteLine("Delivery Receipts:")

            For Each deliveryReceipt As InboundMessageResult In deliveryReceipts

                Console.WriteLine("Code            " + deliveryReceipt.Code)
                Console.WriteLine("Delivery Receipt  " +
deliveryReceipt.DeliveryReceipt.ToString())
                Console.WriteLine("......................

                Console.WriteLine("Text            " + deliveryReceipt.Text)

            Next

        End If
```

PHP example code.

```
$deliveryReceipts = $client->getDeliveryReceipts();

if (count($deliveryReceipts) > 0)
{
   echo "Delivery Receipts: ";

   foreach ($deliveryReceipts as $deliveryReceipt)
   {
      echo "Code: " .$deliveryReceipt->Code. "<br />";
      echo "Delivery Receipt: " .$deliveryReceipt->DeliveryReceipt. "<br />";
      echo "......................

      echo "Text: " .$deliveryReceipt->Text. "<br />";

   }
}
```

Java example code.

```
InboundMessageResult[] deliveryReceipts = client.getDeliveryReceipts();
```

```
                if (deliveryReceipts.length > 0){
                        System.out.println("Delivery Receipts: ");

                        for(InboundMessageResult message : deliveryReceipts){
                          System.out.printf("Code: %s, Id: %d, Source: %s,
Destination: %s",
                                        message.getCode(),
                                        message.getId(),
                                        message.getSource(),
                                        message.getDestination());
                          System.out.printf("Received: %s, SourceId: %d, IsReceipt: %s,
text: %s",
                                        message.getReceived(),
                                        message.getSourceId(),
                                        message.getDeliveryReceipt(),
                                        message.getText());
                          System.out.println("");
                        }
                }
```

Retrieve MO messages?

The same code as above but substitute GetDeliveryReceipts with GetInboundMessages

C# example code.

```
var inboundMessages = client.GetInboundMessages();
```

Visual Basic example code.

```
Dim inboundMessages = client.GetInboundMessages()
```

PHP example code.

```
$inboundMessages = $client->getInboundMessages();
```

Java example code.

```
InboundMessageResult[] inboundMessages = client.getInboundMessages();
```

Register for an inbound address for a Push Service

This service is used to see the MO messages for a given URL per account level.

C# example code.

```
var addresses = new List<string>();
addresses.Add("http://mydomain/Inbound.com");

var registerResults = client.RegisterInbound(addresses.ToArray());
registerResults.ToList().
    ForEach(res => Console.WriteLine("Register Result {0} : {1}", res.Code, res.SubCode));
```

Visual Basic example code.

```
Dim addresses As New List(Of String)
addresses.Add("http://mydomain/Inbound.com")

Dim registerResults = client.RegisterInbound(addresses.ToArray())

For Each registerResult As Result In registerResults
Console.WriteLine("Register Result {0} : {1}", registerResult.Code, registerResult.SubCode)
Next
```

PHP example code.

```
$addresses = array();

$addresses[] = "http://mydomain/Inbound.com";

$registerResults = $client->registerInbound($addresses);

foreach ($registerResults as $registerResult)
{
   echo "Register Result " .$registerResult->Code. " : " .$registerResult->SubCode. "<br>";
}
```

Java example code.

```
ArrayList<String> addresses = new ArrayList<String>();
                addresses.add("http://mydomain/Inbound.com");

                String[] addressesArr = new String[addresses.size()];

                for(int i =0; i <addresses.size(); i++)
                {
                        addressesArr[i] = addresses.get(i);
                }

                Result[] registerResults = client.registerInbound(addressesArr);

                for(int j =0; j <registerResults.length; j++)
                {
                        System.out.printf("Register Result %s, : %s",
                                        registerResults[j].getCode(),
                                        registerResults[j].getSubCode());
```

```
        }
```

How can I use the web service (WSDL) call to get the DLR of my sent message via?

Since the wrapper API is not being utilised, when using the webservice calling SendMessages use the *ProductMessageID* as the unique value for the DLR matching.

Below shows the URL with the *ProductMessageID* shown in bold:

> http://smswebservice1.m-science.com/MScienceSMSWebService.asmx/SendMessages?Destination=44<number>&Message=Product%20message%20of%20150816&AccountID=<account>&Password=<password>&DeliveryReceipt=1&Encrypt=false&**ProductMessageID=150816**&SourceAddress=TestWS&SourceTag=100

How do I Encrypt the password?

To prevent the AccountID and Password from being in readable form, they can be encrypted. To send in an encrypted username or password, simply take each character value for strings representing both and add one. This is sufficient to render known words as garbled text. If the encrypted flag is set on a method call, the reverse of this is done inside the server software.

An example function might be:

C# example code.

```
private static void EncryptCredentials(string accountID, string password, out string
encAccountID, out string encPassword)
{
// First encrypt the account id
 char[] procArr = accountID.ToCharArray();
 char[] output = new char[procArr.Length];
 for(int i = 0; i < accountID.Length; i++)
  output[i]= (char)(procArr[i]+1);
 encAccountID = new string(output);

// Encrypt the password

 procArr = password.ToCharArray();
 output = new char[procArr.Length];
 for(int i = 0; i < password.Length; i++)
  output[i]= (char)(procArr[i]+1);
 encPassword = new string(output);
}
```

Visual Basic example code.

```
Private Sub EncryptCredentials(accountID As String, password As String)

    'First encrypt the account id
    Dim procArr As Char() = accountID.ToCharArray()
    Dim output(procArr.Length - 1) As Char
    For i = 0 To accountID.Length - 1
       output(i) = ChrW(Convert.ToInt32(procArr(i)) + 1)
    Next
    Dim encAccountID As New String(output)
    accountID = encAccountID

    'Encrypt the password
    Dim procArrPassowrd As Char() = password.ToCharArray()
    Dim outputPassword(procArrPassowrd.Length - 1) As Char
    For j = 0 To password.Length - 1
       outputPassword(j) = ChrW(Convert.ToInt32(procArrPassowrd(j)) + 1)
    Next
    Dim encPassword As New String(outputPassword)
    password = encPassword

  End Sub
```

PHP example code.

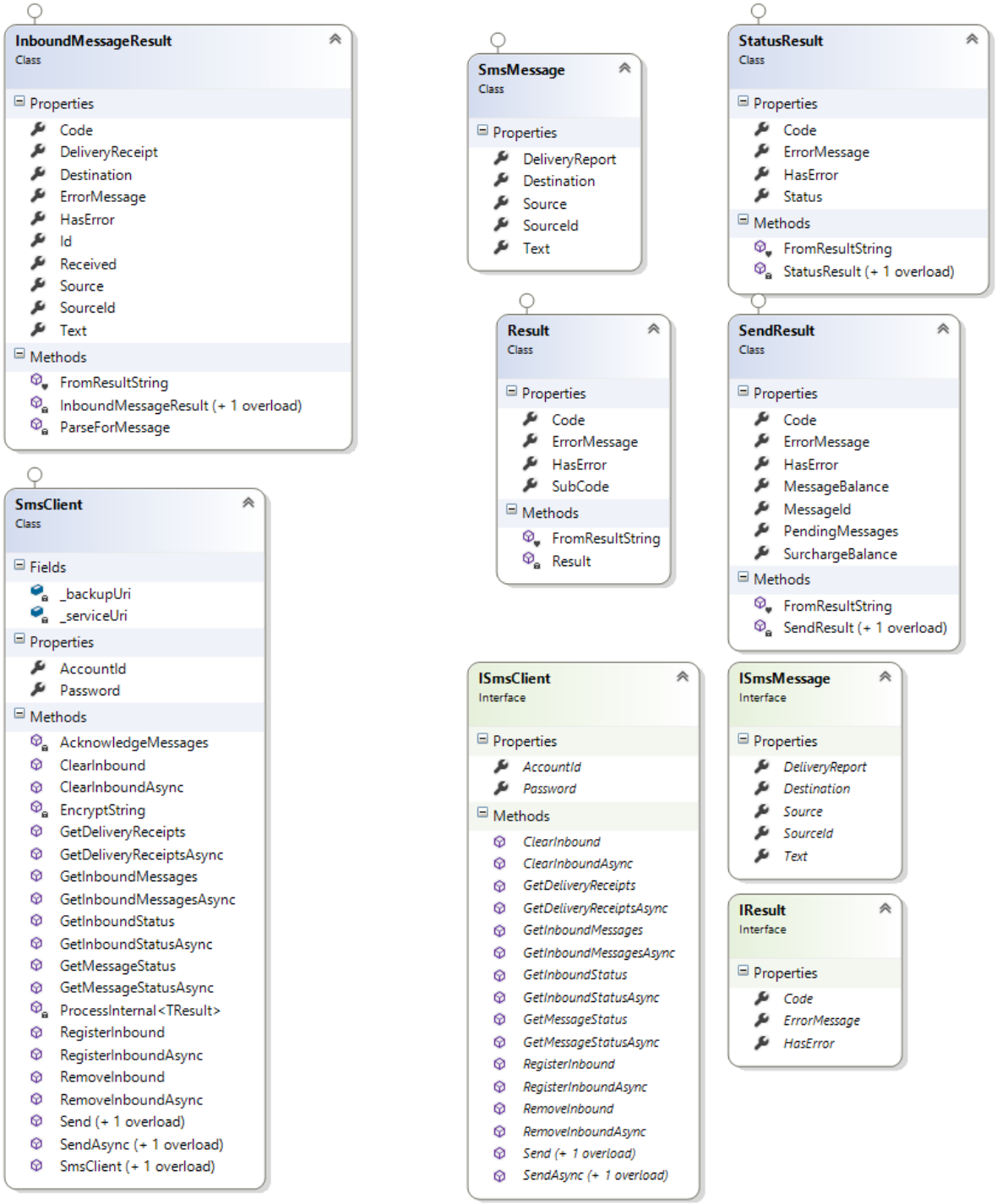
```
function EncryptCredentials($accountID, $password)
{
   //First encrypt the account id

   $proArrChar = str_split($accountID);
   $outputArr = array();
   for ($i = 0; $i < strlen($accountID); $i++)
   {
      $outputArr[] = chr(ord($proArrChar[$i]) + 1);
   }
   $encAccountID = implode($outputArr);

   // Encrypt the password

   $procArr = str_split($password);
   $output = array();
   for ($i = 0; $i < strlen($password); $i++)
   {
      $output[] = chr(ord($procArr[$i]) + 1);
   }
```

```
    $encPassword = implode($output);

    return array($encAccountID, $encPassword);
}
```

Java example code.

```
public static String[] EncryptCredentials(String accountId, String password)
        {
                String[] encryptedArr = new String[2];

                //First encrypt the account id
                char[] procArr  = accountId.toCharArray();
                char[] output = new char[procArr.length];
                for(int i = 0; i < accountId.length(); i++)
                 output[i]= (char)(procArr[i]+1);
                String encAccountId = "";
                for(int j = 0; j < output.length; j++)
                encAccountId = encAccountId + output[j];
                encryptedArr[0] = encAccountId;

                //Encrypt the password
                 procArr = password.toCharArray();
                 output = new char[procArr.length];
                 for(int i = 0; i < password.length(); i++)
                 output[i]= (char)(procArr[i]+1);
                 String encPassword = "";
                 for(int j = 0; j < output.length; j++)
                  encPassword = encPassword + output[j];
                 encryptedArr[1] = encPassword;

                return encryptedArr;
        }
```

SMSClient class and associated classes



DEVELOPING YOUR OWN WRAPPERS

If you wish to write your own wrappers for another language using the WSDL please see this section. Please also contact M Science for further assistance.

Please consider the following in your design.

Fail over Your wrapper should be coded to use the backup address if the primary is unavailable. To utilise secure sockets communications simply prefix the address with 'https://' instead of 'http://'.

Sending a message via HTTP: To send a message, try this simple http send via your browser substituting the Destination, AccountID and Password with your details:

> http://smswebservice1.m-science.com/MScienceSMSWebService.asmx/SendMessages?Destination=44xxxxxxxxx&Message=Hello%20Dragonfly%20SMS&AccountID=xxxxx&Password=xxxxxxx&DeliveryReceipt=0&Encrypt=false&ProductMessageID=123456&SourceAddress=DTTest&SourceTag=0

Message Status The purpose of this function is to determine the message status against the M:Science Host.

The status of a message can be obtained by making a call onto 'GetMessageStatus', which will accept up to ten message IDs and then return a status for each one.

See the 'Function References' section in this guide for full details of the message states which can be returned.

To fully track a message, it is recommended to request a delivery receipt when the original message is sent. If the client needs to interpret delivery receipts directly, the product message ID code can also be supplied with the original message. This is sent back with any delivery receipt and can hence be used to tie the two together.

Receiving Messages If a selected inbound number has been configured to return messages to the web service, the messages can be retrieved by making calls onto 'GetMessages'. This will return up to the next five messages in the queue. To receive more messages, these must first be acknowledged. This can be done by making a call onto 'AckMessages', supplying the list of identity codes given back in 'GetMessages'.

Asynchronous Web Push Service This service asynchronously routes incoming messages via HTTP PUSH to one or more registered URLs. A web address must be provided which interprets the parameter list specified for 'ReceiveInboundMessage'.

The web server code behind the URLs must be capable of accepting an HTTP PUSH (the same as a form submit) using the parameters specified in the 'Client Interface section of this guide.

The user must first contact M:Science at enquiries@m-science.com to activate their account for this purpose and then must register one or more URLs using ‘RegisterForInbound’.