2007

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

Communication Between Outlook Mobile Services and Mobile Devices

By

Shaun Border

University of North Carolina Wilmington

Wilmington, NC

November 7, 2007

Capstone Committee Members:
Dr. Ron Vetter (chair)
Dr. Jeff Brown
Dr. Ulku Yaylacicegi

Capstone project submitted as a portion of the requirements for the Master of Science Degree in Computer Science and Information Systems

Table of Contents

1. Introduction

The Short Message Service (SMS), also known as text messaging, is evolving from a fun way of communicating to a widely accepted method of business communication. SMS messages are made up of header information and a series of characters not to exceed 160 characters. Although the popularity of text messaging is well established in many countries, in others, such as the US, interest in the thumb-driven phenomenon has only recently skyrocketed. It is evident from data reporting SMS trends that text messaging is becoming a widely used communication mechanism. Consider, for example, these statistics from the Cellular Telecommunications and Internet Association (CTIA), the international association for the wireless telecommunications industry:

- In 2000, 14.4 million text messages were sent per month; by June 2007, the number had increased to 28.8 billion per month (this represents a 130% increase over June 2006).
- In the second quarter of 2007, Verizon Wireless alone says it handled 28.4 billion text messages.

The wireless industry boasted an estimated 233 million subscribers at the end of 2006 in the United States alone and it is estimated that greater than 76 percent of the United States population uses wireless services [10]. During the second half of 2006 this industry had total revenue of 65 billion dollars and 8.7 billion of that is due to wireless data revenue, this is up 12.1 percent and 82 percent respectively over 2005 [10]. It is evident that the wireless

industry is huge and growing, so it makes sense to investigate utilizing this technology to bridge the communications gap between personal computers and mobile devices.

Microsoft recognized the text-messaging phenomenon as an up and coming technology for communication with the release of Outlook 2007. That is, Microsoft added a component to allow communication between Outlook 2007 and cellular devices via SMS. This component is known as Outlook Mobile Service (OMS). This component gives Outlook the ability to become a complete personal information manager and offers users the ability to integrate various mobile devices with Outlook without the need for additional specialized software or hardware.

To satisfy the capstone portion for the Masters of Computer Science and Information Systems degree, a software development project was undertaken to design and implement a system to enable communication between the commonly used email application, Microsoft Outlook, and the popular cell phone application, text messaging. Additionally, an in-depth investigation was carried out to determine how this technology is best used for things such as Outlook's calendar and appointment alert features. The project was implemented in collaboration with Mobile Education LLC, a mobile services content provider. Mobile Education provided access to the specific SMS technology that permitted access to commercial cellular providers' networks.

The system implementation for this project consists of the following three parts:

1. OMS Component: This is a web service that is called directly within Outlook 2007 to send text messages.

2. Mobile Education Interface: An interface to provide the technology required to gain access to cellular providers' networks, so text messages can be delivered to mobile devices.
3. OMS Front End: The front end is a web application where potential OMS users can register for service, and customize preferences.

Project deliverables included a functional OMS system that:

a. Sends messages to mobile devices
b. Tracks usage statistics
c. Authenticates user credentials
d. Provides a means for user registration
e. Allows users to customize settings for sending and receiving SMS messages
f. All documentation associated with system design
g. An analysis detailing the possible uses for this technology

Figure 1 depicts the general SMS infrastructure for processing SMS messages. Rather than communicating directly with the various short message service centers (SMSCs), content providers often go through a message aggregator instead. The message aggregator uses the Short Message Peer-to-Peer Protocol (SMPP) to maintain connections with carrier networks. The OMS web service component developed for this project interfaces with the SMS infrastructure via an application programming interface (API) residing on Mobile Education's content server. So, when an Outlook user sends a message, it is sent via the web service through Mobile Education's content server which then passes it on to the SMS broker

(aggregator) and thereby reaching the cell carrier network and ultimately the cell phone to which it is destined.

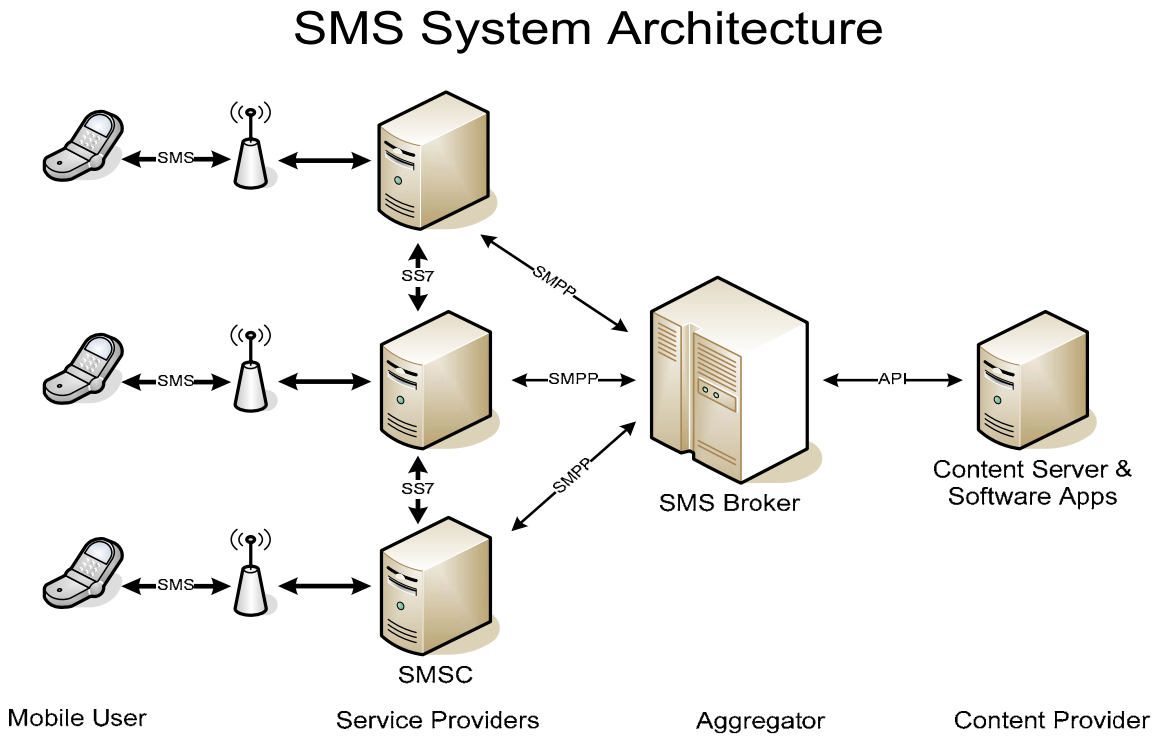


Figure 1 – The general system architecture for processing SMS messages

After the communication between an Outlook client and a cell phone was established, an investigation into new and innovative methods for utilizing this technology was explored. One particular application area that was considered involved looking at how this technology could be used for rapid mass communications with mobile devices through distribution lists within Outlook. This functionality can be used to notify large groups of people directly to their mobile devices during emergencies, where time is critical. Next, we focused on establishing a bi-directional communication channel between Outlook clients and mobile devices. That is, when a text message is sent from Outlook and received on a cell phone, the user of the phone can reply to the text message and the reply will be sent back to Outlook and

appears in the sender's email inbox. Finally, the sending of Calendar Appointments and alerts to mobile devices from Outlook was setup and studied.

2. Outlook Mobile Service

The Outlook Mobile Service component works by utilizing three stub functions that are accessed via a web service. A web service was developed that fulfills the functionality of the three stub functions.

2.1 GetServiceInfo

GetServiceInfo() is one of the stub functions. Outlook calls this function to retrieve properties, such as, authentication types, supported services and parameters of supported services of the OMS service. The Outlook client calls GetServiceInfo() to configure it's self for the web service specified.

2.2 GetUserInfo

Next is the GetUserInfo(string xmsUser) function. This function is used to retrieve information about the user sending the text message. This function retrieves the reply mobile number, email address, and error code. This function accepts an XML formatted string that contains the user's authentication information.

2.3 SendXms

Finally, the SendXms(string xmsData) function is responsible for sending the SMS message to the mobile device. The xmsData parameter is an XML formatted string that contains all the header and message information that is required to send the message to the

target mobile user. Once that portion is complete, Outlook is able to successfully send and receive text messages.

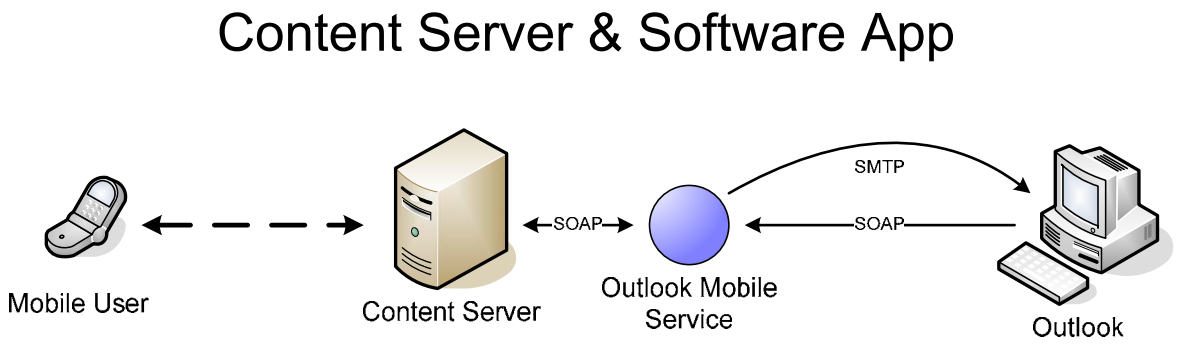


Figure 2 – OMS Access to Content Server and Cellular Network

Figure 2 shows how an Outlook user calls the OMS web service via the Simple Object Access Protocol (SOAP) to access the cell phone service provider's network through another SOAP interface to Mobile Education's content server. Note that the OMS service is able to relay SMS messages originating from a mobile device to the Outlook client via the SMTP protocol.

Another view of the components in the system is shown in Figure 3. This figure also depicts the OMS registration system which authenticates user credentials and provides a means for user registration.

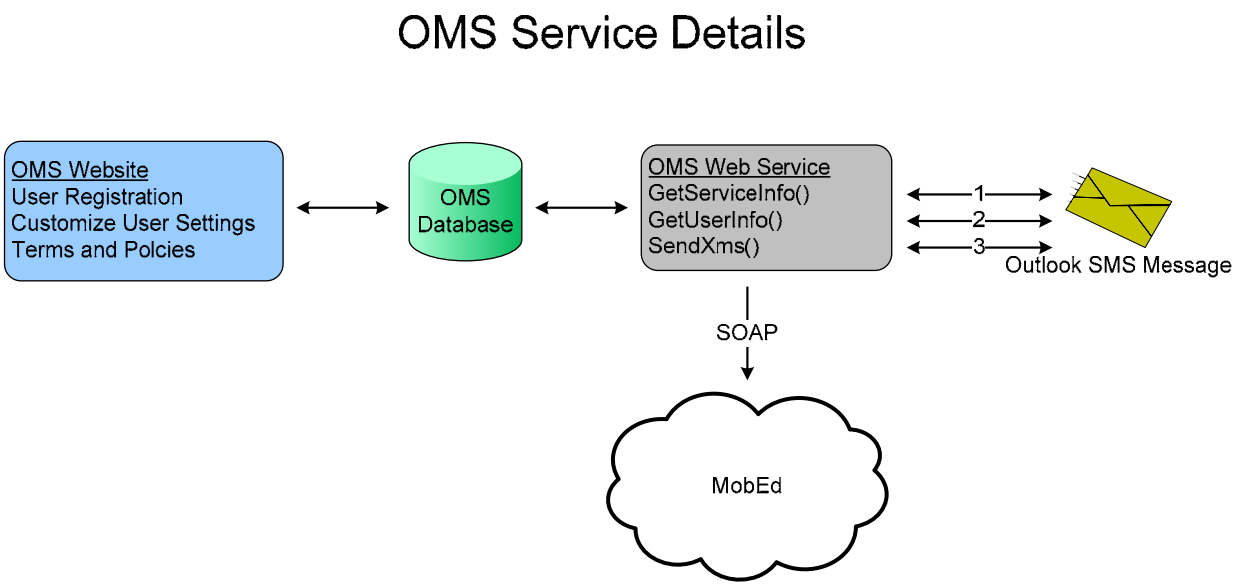


Figure 3 – OMS Service Details

3. Analysis and Design

The project was designed employing industry standard software design practices. The software development design lifecycle used to implement this project was based on the iterative model. This approach was utilized because it allows the project to be implemented and delivered in phases, thus providing an opportunity to assess the product after each delivery phase to uncover potential enhancements that could be added during the next phase of development. Since the Outlook Mobile Service (OMS) is a new feature, there is not an abundance of information or resources available discussing designing and implementing OMS. Indeed, some of the information was even incorrect. Therefore, to use some of the more static software development lifecycles, such as the waterfall model, was not an option.

3.1 Technologies and Architecture

Microsoft C# and the 2.0 .Net framework were used as the primary language and programming environment for implementing the project. The main reason for these choices was due to the fact that the Outlook Mobile Services specification states that the OMS server must run on Microsoft's Internet Information Server (IIS) server and the IIS server by default is, configured to render applications written using the .Net framework, not Java. Additionally, Microsoft SQL Server 2005 was chosen for all required database transactions. SQL Server 2005 offers XML integration which enables "The storage of XML fragments or documents" [8]. It also provides better integration with Visual Studio over Oracle's 10g database. As stated by Ruebush, "SQL Server integration is more comprehensive, more seamless, more functional, and better performing than Oracle 10g" [3]. SQL Server 2005 also provides native support for web services by allowing external applications to "invoke stored procedures, Transact-SQL statements, and user-defined functions" [3]. Windows 2003 Server running IIS 6.0 was used to host the developed application. The reason for this selection is that IIS is required to run .Net applications and Windows 2000 is running IIS 5.0, which does not fully utilize the .Net framework.

3.2 Requirements

The requirements for the project were collected through a variety of methods. For instance, some of the requirements were gathered by reading the Microsoft documentation for implementing an Outlook Mobile Services server. Other requirements were collected in discussions with Mobile Education LLC. The last phase of requirements was collected by discussing the project with outside industry professionals. Refer to Appendix A for detailed system requirements.

3.3 Use Cases

3.3.1 Account Setup

Bob decides he needs to communicate with mobile devices directly from Outlook, so Bob goes to oms-uncw.homeip.net and registers for an account. Bob is required to specify a username, password, email address, mobile number, first name and last name. Additionally, Bob can define optionally specify a middle name, reply to mobile, reply to email, max messages sent per day. After clicking save Bob's information is committed to, the database and he can now setup Outlook to use his account credentials and start sending SMS messages. Bob opens up Outlook and selects Tools from the menu and Account Settings. Next, he clicks New to add a new account. He clicks the Other radio button and selects Outlook Mobile Service (Text Messaging). Bob then enters the OMS service URL and inputs his username and password. Next, he clicks next and is instructed to close and reopen Outlook. After Bob reopens Outlook, he is able to send text messages.

3.3.2 Send SMS Message

Sue decides to send an SMS message from Outlook. She opens Outlook and clicks on File and New then selects Text Message. She specifies who the recipient(s) of the message will be by inputting their mobile numbers including area code. Next, Sue enters the message she wants to send and clicks the send button. If an error occurs during message, processing a message appears in Sue's inbox notifying her and detailing the error specifics.

3.4 Database Design

This project did not require an extensive database to be developed. The primary function of the database within this system is to store user specific information. Therefore, it

was implemented using standard database design practices and achieves all criteria specified by Boyce-Codd normal form. See Appendix B for an Entity-Relationship (ER) diagram.

3.4.1 OMS_USER Table

The primary table used by the system is the oms_user table that is where all the user information and preferences are stored.

3.4.2 MESSAGE_LOG Table

The message_log table is used to store exceptions that occur during the processing of SMS messages. Since this system runs primarily as a backend service, it could become very difficult to identify where and the type of errors that occur within the system. The message log table is used to log when messages are sent and who sent them. This table also has a routing number which can be used to determine the sending user of a message originating from an Outlook client, so if the message recipient reply's to the message it can be routed properly back to the end user.

3.4.3 SERVICE_PROVIDER Table

The service_provider table contains a list of mobile service providers that can receive messages sent from the OMS client, this table is also used to take a service provider name and lookup their corresponding client id which is then used to properly deliver the SMS message.

3.4.4 PHONE_NUMBER Table

Finally, the phone_number table stores the phone numbers that a particular OMS user has sent messages to and has a pointer to the service provider's client id. Prior to sending a

message to a mobile device through Mobile Education, the recipient's service provider's client id is required. MBLOX, the aggregator used by Mobile Education to send SMS messages, does not provide any mechanism for looking up carrier ids from a mobile number. Therefore, this functionality was built into the OMS service, using a public web page to do a reverse lookup of a mobile number to determine the service provider name. Once the provider is determined, that information is stored in the database's phone number table so that a reverse lookup only need to occur once instead of each time that mobile number is the recipient of an OMS message. One problem with the reverse lookup system is that it does not account for mobile numbers that may have been transferred to a different service provider from the original number assignment. Therefore, the mobile number table will also allow OMS users to manually specify the service provider for each number in the OMS user's phone number list. Additional design considerations to solve this problem involve (a) using another aggregator (there are other aggregators that provide services to do carrier id lookups based on phone number) or (b) require OMS users to build an address book of mobile numbers and only allow users to send messages to the predefined list of mobile numbers. The predefined list is not a viable option since it would require entirely too much setup and unneeded configuration on the user end. Using a different aggregator is probably the best option, however since the OMS service is using Mobile Education's messaging platform, this project did not have a choice in which aggregator to use. Doing a reverse lookup from a public web site was determined to be a good compromise between the two options discussed above. It allows users to send SMS messages and compile a phone book for the user automatically for those phone numbers for which a carrier id was found, while at the same

time giving the user the ability to customize the phone book by changing the service provider as needed.

4. Implementation

The web service design for this system was done in a manner that emphasizes code reuse and maintainability. By modularizing functionality, system maintenance and bug diagnosis time is greatly reduced. Therefore, the web service was designed by dividing common functionality and grouping these into classes or objects. Four classes are used to handle all the processing required to validate a user, build an SMS message, retrieve service settings and send SMS messages to Mobile Education for delivery. The classes are OMSUser, OMSMessage, MobedInterface, and Utility. The web service is made up of three web methods defined by Microsoft that are required to enable Outlook to send SMS messages – they are GetServiceInfo, GetUserInfo, and SendXms. Additionally, a web method, named SMStoEmail, is in the web service that will accept incoming messages from Mobile Education and route these messages to a user's Outlook inbox. The subsequent sections will first discuss the individual classes and functionality of each. Later, we will detail how each web method uses these functions to process and send SMS messages.

4.1 Classes

4.1.1 OMSUser Class

The OMSUser class is used to load user information from the database, parse and extract user credentials from an XML document, and get usage statistics for a user. It has the following two constructors:

- public OMSUser(String xmsUser)
- public OMSUser(String _userID, String _password)

The first constructor takes a string parameter named xmsUser which is an XML document that contians the username and password for the current user. This information is extracted from the document and used to obtain the remaining information related to the current user. The second constructor takes a _userID string and _password string that are authenticated to verify user valiadity and obtain the remaining informaiton related to the current user.

Once an OMSUser object is created it will has access to the following procedures:

- public Boolean validateUser() - The validateUser function is used to authenticate the user account for the current user object. It uses the username and password either provided through one of the constructors or from the parsing of the xmsUser XML document that was passed in through the other constructor.
- public String getXMLString() - The getXMLString function is used to build an XML document that conforms to the xmsUser XSD defined in appendix D.
- public int getDaysMessages() - Function getDaysMessages is used to determine the number of messages that the current user has sent today.
- public int maxMessagesPerDay() - Function maxMessagesPerDay retrieves the maximum number of messages a user can send in one day.
- public int getU_ID() - Function getU_ID is used to retrieve the primary key value of the oms_user table for the current user.

4.1.2 OMSMessage Class

The OMSMessasge class is used to parse an xmsData XML document, build the SMS message and send the message to Mobile Education for delivery. It has one constructor, which is defined as follows:

- public OMSMessage(String xmsData) - The OMSMessage takes a string parameter named xmsData, this parameter will have all the message information required to send an SMS message. See appendix D for xmsData schema. The OMSMessage constructor validates the xmsData parameter against the schema definition to ensure validity. This parses the parameter to extract the data within the XML document and stores them into parameters that can be used by other parts of the program.

Once the OMSMessage object is created, it has access to the following procedures:

- public String sendMessage() - Function sendMessage takes the the current instance of the OMSMessage and sends it to Mobile Education for delivery. This function creates an instance of the OMSUser class and validates the user creditentials to ensue that the message is being sent from a valid user. In the event of an invalid user and error is generated and passed back to the Outlook client. It then checks to see if by sending this message will cause the user to exceede the max number of messages per day defined in the user preferences. If it will exceede the max messages per day value then an error is created and returned to the Outlook client. Next this function calls the MobedInterface to send the message to the mobile device.

4.1.3 MobedInterface Class

The MobedInterface class is used to send messages to Mobile Education for delivery. This class also has code in it to do reverse lookups on mobile numbers to determine the service provider and subsequentily their corrsponding client id. The MobedInterface is a static class and can not be instancited, meaning it does not have a constructuor. The publicly available functions are as follows:

- public static Boolean sendSMS(String[] recipientList, int numberOfMsg, String[,] message, int u_id) - The sendSMS function is used to create and send the SMS message takes a the following list of parametrs a string array of the recipients the sms message is going to be deliverred to, numberOfMsg is an int, a two deminsional string array of the message, and an int called u_id which is the primary key from the oms_user table. The numberOfMsg parameter details how many SMS messages are going to be sent. In the event that an SMS message exceeds the maximum characters defined for an SMS message by the OMS service, the message will be split into multible messages and the numberOfMsg paramer details this. The two deminsional array of message contains the type of message, MIME type and the message body. The sendSMS function first attempts to get the carrier code by looking in the phonenumber table see if this u_id has send an SMS message to the recipient. If a record is found in this table the service provider's carrier ID is returned if not, the system queries whitepages.com to execute a reverse lookup to determine the service provider for the current mobile number. Once the service provider is determined the recipient phone number and carrier ID is saved in the phone_number table so it can be

retrieved again if the the current user resends to this recipient without the need to execute another reverse lookup to determine the carier ID.

4.1.4 Utility Class

The Utility is a static non instantiable class, it provides various functions that are commonly used throughout the application. The following is a list of the functions in the Utility class:

- public static String LoadXmlFromFile(string strFile) - The LoadXmlFromFile function reads the contents of the XML file pointed to by the strFile parameter and returns its contents to the calling procedure.
- public static Boolean ValidateXMLString(String xmlDoc, String xsdPath) - The ValidateXMLString function validates the XML document in the xmlDoc parameter against the schema definition file pointed to by the xsdPath. This function returns a Boolean value that indicates weather the xmlDoc is valid against the schema definition file.
- public static String cleanString(Object str) - The cleanString function takes and object and converts it into a string, this is used when extracting data from a database and storing in local variables.
- public static void SaveLogFile(string strErr, string strFileName) - SaveLogFile will save a file of the name in strFileName and add the contents of strErr. This was primarily used during early development to help trace program workflow.

- public static SqlConnection getDBConnection() - The getDBConection function creates a database connection object and returns it. Any time the application needs to interact with the database it calls this function to create it.
- public static Boolean isValidUser(String userName, String password) - The isValiduser function validates the user credentials passed in as parameters against the database to see if a valid user was passed in or not.
- public static void BuildError(XmlTextWriter wr, string errCode, bool bFailed, string strContent, string strRecipients) - The BuildError function is used to create an XML document that contains an error message that needs to be returned to the Outlook client.
- public static void logError(int u_id, String message) and public static void logError(String message) - The logError function is overloaded, it saves a record in the application_log table when an unexpected exception occurs during program execution. This was done so that if a user reports an error, developers can easily go to the application log table to get additional information.

4.2 Web Methods

4.2.1 GetServiceInfo

The GetServiceInfo web method is used to retrieve settings of the OMS server. Some settings that can be defined as server settings are serviceUri, localName, authenticationType, and serviceType. The signature of this web method:

```
[System.Web.Services.WebMethod()]
public string GetServiceInfo()
```

The XML document returned by this function for this implementation of the OMS service is as follows.

```
<?xml version="1.0" encoding="utf-16"?>
<serviceInfo xmlns="http://schemas.microsoft.com/office/Outlook/2006/OMS">
 <serviceProvider>UNCW OMS</serviceProvider>
 <serviceUri>https://sblaptop/oms-service/oms.asmx</serviceUri>
 <signUpPage>https://oms-uncw.homeip.net/signup</signUpPage>
 <localName>OMS UNCW</localName>
 <englishName>OMS UNCW</englishName>
  <authenticationType>Other</authenticationType>
  <supportedService>
     <SMS_SENDER maxRecipientsPerMessage="100" maxMessagesPerSend="20"
       maxSbcsPerMessage="140" maxDbcsPerMessage="70" />
  </supportedService>
  </serviceInfo>
```

This function is called when a user setups an OMS client, Outlook then stores these values and does not recall this method again.

4.2.2 GetUserInfo

This is the next method called from Outlook when setting up an OMS client. This function takes in an XML document that contains the username and password for the account that is being setup and returns additional information about the user in the form of the userInfo schema definition file displayed in the appendix of this document. The signature of this web method:

```
[System.Web.Services.WebMethod()]
public string GetUserInfo(String xmsUser)
```

A sample of the XML document returned by this method is:

```
<userInfo xmlns="http://schemas.microsoft.com/office/Outlook/2006/OMS">
<replyPhone>9102007072 </replyPhone>
<smtpAddress>demo@gmail.com</smtpAddress>
<error code="ok" />
</userInfo>
```

4.2.3 SendXms

The SendXms web method is called each time a user sends an SMS message from the Outlook client. The SendXms web method takes an XML document that contains the sending user's credentials, recipient information and message body. The signature for this web method is:

```
[System.Web.Services.WebMethod()]
public string SendXms(String xmsData)
```

A sample of the of the xmsData input parameter is:

```
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
<xmsData client="Microsoft Office Outlook 12.0"
xmlns="http://schemas.microsoft.com/office/Outlook/2006/OMS">
<user>
        <userId>testuser</userId>
        <password>samplepass</password>
        <replyPhone>9102007072 </replyPhone>
        <customData/>
</user>
<xmsHead>
        <requiredService>SMS_SENDER</requiredService>
        <sourceType>xmsInspector</sourceType>
        <to><recipient>9102007072</recipient></to>
</xmsHead>
<xmsBody format="SMS">
        <content contentType="text/plain"
        contentId="Att0.txt@AB1B43B2B0594564.B94EF7ABB12B49BA"
        contentLocation="1.txt">Test Message</content>
</xmsBody>
</xmsData>
```

4.2.4 SMStoEmail

The SMStoEmail web method is used to accept incoming SMS messages and route them to the corresponding OMS user's inbox. When Mobile Education received an SMS message that begins with OMS <*Mobile Number*> it will send that information over to the

OMS service for routing and delivery. The mobile number that follows OMS in the SMS message body is used to determine the recipient of the SMS message. The signature for this web method is:

- public string SMStoEmail(String from, String message)

This method has two parameters, the first is the sending users mobile phone number. The phone number can be in the form of country code, area code, and mobile number (e.g., 0011115555555) or it can be in the form of area code, and mobile number (e.g., 1115555555). The second input parameter is the message body which as formentioned will begin with OMS <*Mobile Number*>. The mobile number is extracted from the message body and used to lookup the email address of that mobile number. If no user if found with that mobile number then the message is dropped and not delivered.

5. Discussion

During the course of completing this project several decisions had to be made with respect to how to approach and overcome problems that arose during the design and implementation of the system. This section discusses some of the problems encountered and the solutions and reasoning behind the ultimate approach taken. In addition, we discuss the use of Outlook Mobile Services for mass communication efforts and receiving calendar updates. In general, the issues can be divided into the following categories: (a) type of message delivery, (b) user authentication, (c) enabling two-way communication, (d) database interaction, and (e) how to lookup the client ids for service providers.

5.1 Message Delivery

The type of message delivery was one of the first decisions that needed to be made, because it is directly related to the core functionality of the system. There are basically two types of message delivery techniques for delivering messages over networks, which are, best effort delivery and guaranteed delivery. Best effort delivery does exactly what the name says – it makes its best effort to deliver a message to its intended recipient. This technique for transferring data does not have a guarantee, so in the event that the message does not reach its intended recipient the message is dropped from the network. Guaranteed delivery ensures that data reaches its intended recipient, so if a message gets lost or corrupted a guaranteed delivery system will regenerate the data and automatically send it out again. This project uses the best effort delivery technique. The primary deciding factor in making this decision is that SMS uses the best effort delivery protocol. The amount of work required to build a guaranteed delivery component for this system was seen as unnecessary overhead. Should the data sent across the system be mission critical, meaning lives or businesses could be at stake, then another technique should be sought after and implemented to provide a more reliable delivery protocol for OMS.

5.2 User Authentication

User authentication was another decision that needed to be made during the design of this project. According to Microsoft documentation, Outlook Mobile Services supports two types of Authentication: Passport and Other. These authentication options are specified in the serviceInfo.XML document and described in the serviceInfo schema definition file located in Appendix D. A value of Passport indicates that Microsoft's Passport Network that

is a service that Microsoft provides to allow applications to authenticate users using this passport code. Its advantages are that user credentials do not need to be stored locally in an application and that authentication will be automatically handled for you. One disadvantage is that you have less control over the accounts in your system, and in the event that the Passport system is unavailable, users will be unable to use system. An alternative means of authentication for Outlook Mobile Services is to build custom authentication into the application. For this project implementation, the custom authentication method was selected for the following reasons. First, all traffic sent between the Outlook client and the OMS service will be encrypted because of the SSL certificates. The difficulties in supporting such an approach is that it can become a help desk burden, because you will be required to field support calls for users that have forgotten their password.

5.3 Bi-directional Communication

Enabling two-way communication between the Outlook client and the mobile device was one of the more challenging tasks to address in this project. There were several factors that needed to be taken into consideration when designing and developing this functionality. One possible technique for implementing this solution would be to interact with a mobile service provider directly instead of through the MBLOX aggregator or to change to an aggregator that provided more robust functionality. This would have allowed us to use the actually senders mobile number as the from address when sending from the Outlook client. Then, once the SMS message was delivered, the user could simply reply to the message and it would be delivered to the mobile device of the original sender. The main problem with this approach is that, there is no technique for getting the reply message back into the Outlook client. In order to achieve that you would need a mechanism for intercepting SMS messages

as they come into the service providers before they are sent out for delivery. This was not an option using the SMS infrastructure that was made available by Mobile Education.

When the SMS messages are not intercepted before they are sent for delivery, then because the from field in the original sender's message is replaced by the content provider's short code there is no way for the recipient cell phone to reply and have it go back to the sender. Thus, the very first message would be sent from the Outlook client and all subsequent messages and replies would be between the mobile device and the content provider's short code.

The next option considered appeared at first glance to be a "patchy" fix but after further investigation, it offered a lot more than it originally appeared. This option consists of sending messages from the Outlook client as usual and if the user with the mobile device decided to reply they just being the message with "OMS <*Dest Mobile Number*>". As stated earlier, since all messages sent through Mobile Education are delivered with the short code 90947, all message replies will also go back to 90947. Fortunately, adding "OMS <*Dest Mobile Number*>" also gave us a means for intercepting SMS messages and routing them back to the Outlook client and it did not require any additional coding. Thus, it turned out to be a reasonable solution to the problem.

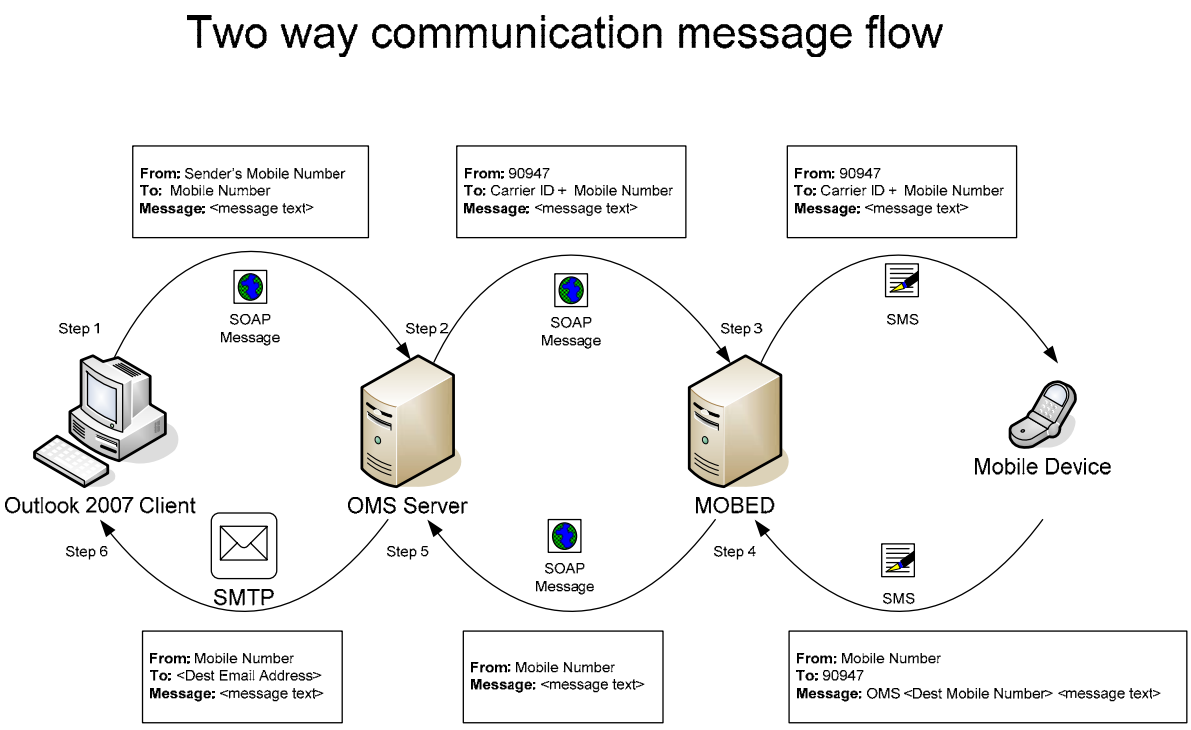


Figure 4 – Two-way communication message flow

5.4 Database Interaction

There are several techniques employed in the corporate word for interacting with databases. The primary technique used for interacting with databases is to use inline queries and execute the queries from the application. This method of interacting with a database has certain security risks if the database is not setup correctly. Users can prematurely terminate queries and enter in an unintended command to execute. If a user does this, they could do all types of malicious activity to the database. Another technique for interacting with databases is to only allow the application to interact with store procedures and user defined functions. Employing this technique requires that parameters be passed to the stored procedure or function to dynamically control the data modified or returned to the calling application.

From an implementation standpoint, it is usually quicker to build the application using inline queries, but this technique can actually become more difficult to maintain if future modifications are required. To use the stored procedure or user defined function technique initial development takes a little longer, but it better positions the application to accept future modifications. This application was built to employ the stored procedure and user defined function technique. The reason for this decision was primarily to remove the security risk of allowing queries to be executed from within the application. The increased maintainability is an added bonus with using this technique and even though this application was specked out very well and dealing with indecisive stakeholders was not as issue, the increased maintainability was utilized.

5.4 Sending SMS Messages

The last decision that was made involved how to send SMS messages. The aggregator that Mobile Education uses (MBLOX) to send and receive SMS messages requires that the service provider client id for the recipient of the SMS message be specified for each message sent. With this limitation, a solution was needed to determine the client id (carrier id) for a given phone number. The options considered when working on this problem were to force the OMS user to pre-construct a list of mobile numbers that they intend to send SMS messages to, and have the user specify each mobile number's client id. The other option is to only allow the OMS user to send SMS messages to their personal mobile device. The final choice was to use a reverse lookup service to lookup he client id for the recipient's phone number. If the first option was implemented it would greatly reduce the usability of the application and make sending SMS messages for new users difficult. The technique for allowing users to only send to their mobile device is a viable option because they can setup

outlook to deliver their calendar and appointment notifications as well as forward email messages meeting certain criteria. The main draw back with is that it greatly reduces the usability of the application. So the option selected for final implementation was to do a reverse lookup to determine the client id. Additionally, a decision was made to add a contact list feature that is automatically built for each OMS user. When an OMS user sends an SMS message to a new phone number a lookup is done to determine that phone number's client id, this is then saved in the OMS user's contact list. The contact list provides several benefits, first, when SMS messages to a number in the contact list the client id can be easily located and does not need to be re-resolved and the OMS user can modify the contact list because the reverse lookup only returns the correct client id for mobile numbers that have not been transferred from the original carrier to which they were assigned. In the event that a mobile number is mapped to an incorrect client id, the OMS user can always go into the system and update the service provider for that particular mobile number.

5.5 Outlook Mobile Services – Mass Communication

Using Outlook Mobile Services for mass communication definitely has advantages, however based on the technologies used to implement the project it would not be recommended. There are two primary reasons for this recommendation. First, is that MBLOX requires that each recipient's client id be provided to successfully send an SMS message. Additionally, MBLOX will only accept a finite number of messages over a short period of time. After this number has been reached, MBLOX will deactivate Mobile Education's ability to send SMS messages. If these limitations did not exist, then SMS messages could be sent without specifying the client id and there would be no maximum on the number of messages sent and then using this service for mass communication would be a

very effective technique for reaching large groups of people. If Mobile Education had an aggregator that did not have these limitations, then another question arises. That is, where should message replication occur? Since an SMS message can only have one recipient per message, a message that needs to be sent to large groups of people must be replicated for each recipient within the group. The options to solve this problem are: (a) to send one message to Mobile Education and have the message replication occur within that system, or (b) replicate the message within OMS and send each SMS message to Mobile Education for delivery. The Outlook Mobile Service is currently setup to handle this type of message replication. However, the downside is depending on the number of recipients, there could be large amounts of data being transferred across Mobile Education's Internet connection and could theoretically diminish Mobile Education's ability to communicate with MBLOX. If the target group were around 1000 contacts, it is recommended that the replication occur within OMS, because neither Mobile Education nor OMS will need to be reconfigured to support this method of replication. If the target group is greater than 1000, then the replication should occur within Mobile Education, but Mobile Education would need to implement a new web service to handle this process.

5.6 Outlook Mobile Services – Calendar Updates and Appointments

Using Outlook Mobile Services to receive calendar updates and appointment notifications was found to be very effective. This allows OMS users to use standard technology that is readily available for receiving this information. There are significant advantages with this technique over synchronizing mobile devices with Outlook. First, OMS users will not need to buy specialized hardware and software designed to connect their mobile device to their PC. In the event that an OMS User gets a new mobile device that has

the same phone number, the user will not need to do anything to continue receiving calendar appointments and notifications. If the user were not an OMS user, that user would need to buy new hardware and software and then spend time installing and configuring this device to work with Outlook. In addition, the user never needs to connect to the PC to receive updates, so for instance, if the OMS user was on a two-week business trip, that user would receive calendar updates and notifications over the entire duration of the trip and the hardware-synchronizing user would not. Finally, executives can have their calendars sent directly to their assistant's mobile devices and this is not available for other methods that use a synchronization technique.

6. Summary and Conclusions

In summary, it was learned that SMS is a viable method for communication with mobile devices. In fact, next to voice calls, SMS messages are the second fastest technique for communicating with a person. Outlook Mobile Services are an excellent technique for sending SMS messages from Outlook to mobile devices. Since this is such a new technology there still is a need for bugs to be worked within the Outlook client, but overall this is a very efficient and effective method of sending SMS messages. As SMS technologies continue being more widely accepted, the OMS service will get more use and become more cost effective.

7. Bibliography

[1] "Apache Axis2 User's Guide - Creating Clients." Axis/Java. Oct. 2007 <http://ws.apache.org/axis2/1_1_1/userguide-creatingclients.html>.

[2] "Google Calendar." Google. Oct. 2007 <http://www.google.com/calendar>.

[3] Ruebush, Mitch. "Comparing SQL Server 2005 and Oracle 10g as a Database Platform for Microsoft .NET Developers." Rev. of Microsoft SQL Server 2005.

[4] Shen, Paul, Bing He, Yong Zhang, and Judy Zhang. "Outlook 2007 Mobile Service Guidelines (Part 1 of 3)." MSDN. Feb. 2007. July 2007 <http://msdn2.microsoft.com/en-us/library/bb277361.aspx>.

[5] Shen, Paul, Bing He, Yong Zhang, and Judy Zhang. "Outlook 2007 Mobile Service Guidelines (Part 2 of 3)." MSDN. Feb. 2007. July 2007 http://msdn2.microsoft.com/en-us/library/bb277363.aspx.

[6] Shen, Paul, Bing He, Yong Zhang, and Judy Zhang. "Outlook 2007 Mobile Service Guidelines (Part 3 of 3)." MSDN. Feb. 2007. July 2007 http://msdn2.microsoft.com/en-us/library/bb277362.aspx.

[7] "Short Message Service." Wikipedia. 6 July 2007. July 2007 <http://en.wikipedia.org/wiki/Short_message_service>.

[8] "SQL Server 2005 Features At a Glance." Microsoft. July 2007 <http://www.microsoft.com/sql/prodinfo/features/features-at-a-glance.mspx>.

[9] "Text.It." Aug. 2007 <http://www.text.it/mediacentre/press_release_list.cfm?thePublicationID=ED01B109-FD55-2155-DB2178BC249826CC>.

[10] "Wireless Quick Facts." CTIA. Dec. 2006. 03 July 2007 <http://www.ctia.org/media/industry_info/index.cfm/AID/10323>.

8. Appendices

8.1 Appendix A - Requirements for Project

#	**Requirement Description**	**Comments**
UR1.01	Implement the GetServiceInfo function. Outlook utilizes this function to gather information about the OMS service during the account registration process.	
UR1.02	Implement the GetUserInfo function. Outlook utilizes this function to gather information about the registered user for the OMS service, this is called during the account registration process.	
UR1.03	Implement the SendXMS function. Outlook utilizes this function to send text messages generated within Outlook to mobile devices.	

#	Requirement Description	Comments
UR1.04	Only users that have registered with the OMS service will be able to send text messages.	
UR1.05	Users must accept terms and agreements before they will be able to utilize the OMS service.	
UR1.06	Users must be authenticated prior to each text message is sent.	
UR1.07	The system must validate all XML against the predefined XSDs defined by Outlook	
UR1.08	The system must be able to correctly handle and route replies to text messages that originate from Outlook.	This is a loose requirement as there are limitations for sending messages through mobile education.
UR1.09		

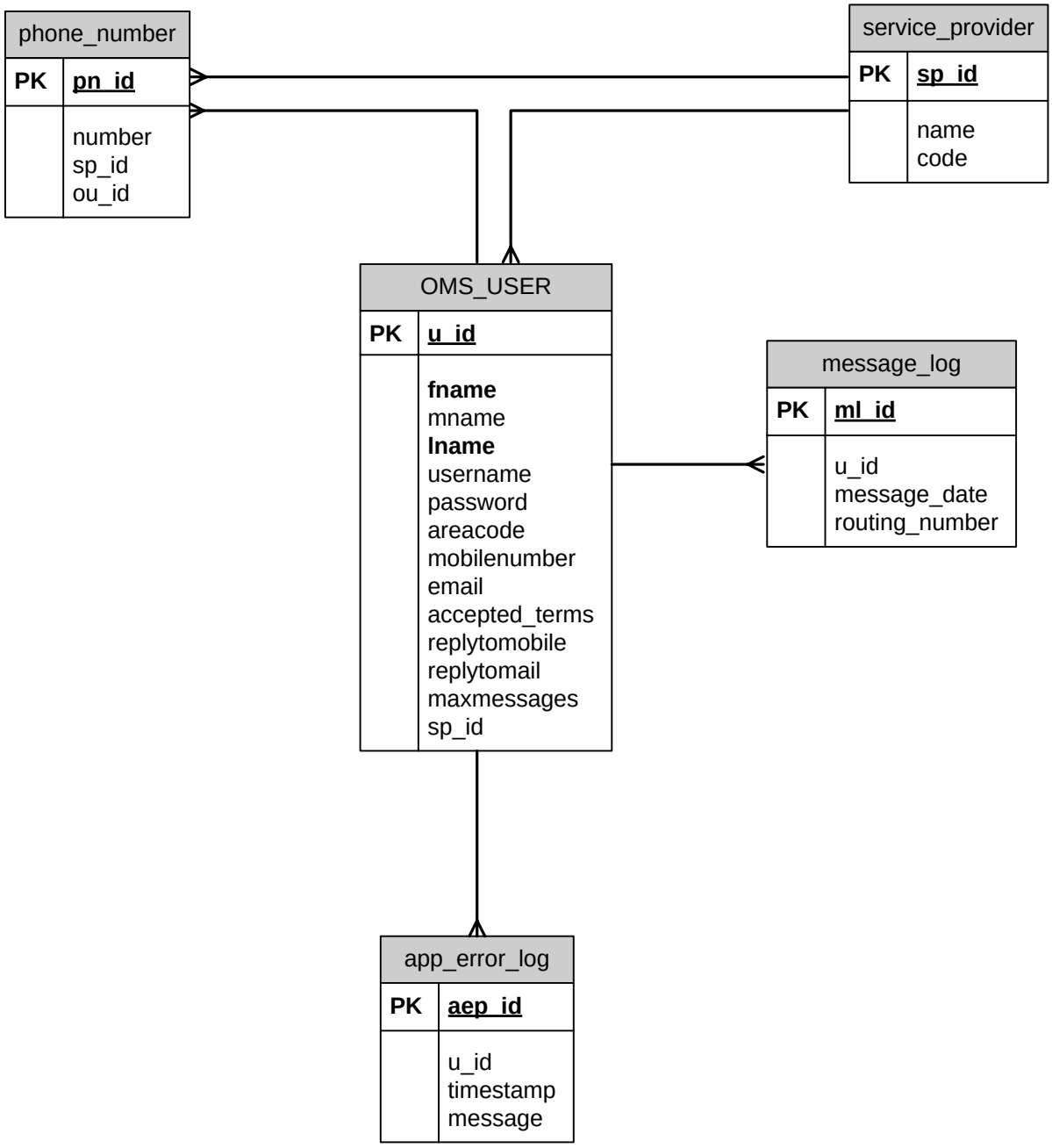
#	Requirement Description	Comments

#	**Requirement Description**	**Comments**
UR2.01	The system must be able to send text messages to Mobile Education for mobile delivery.	
UR2.02	OMS must be able to look cellular providers codes based on common names	
UR2.03	Handle errors that occur when sending messages to Mobile Education	
UR2.04	Mobile Education must be able to relay messages back to OMS for routing when a mobile user responds to a message that originated from OMS.	This is a loose requirement due to limitations of Mobile Education's technologies

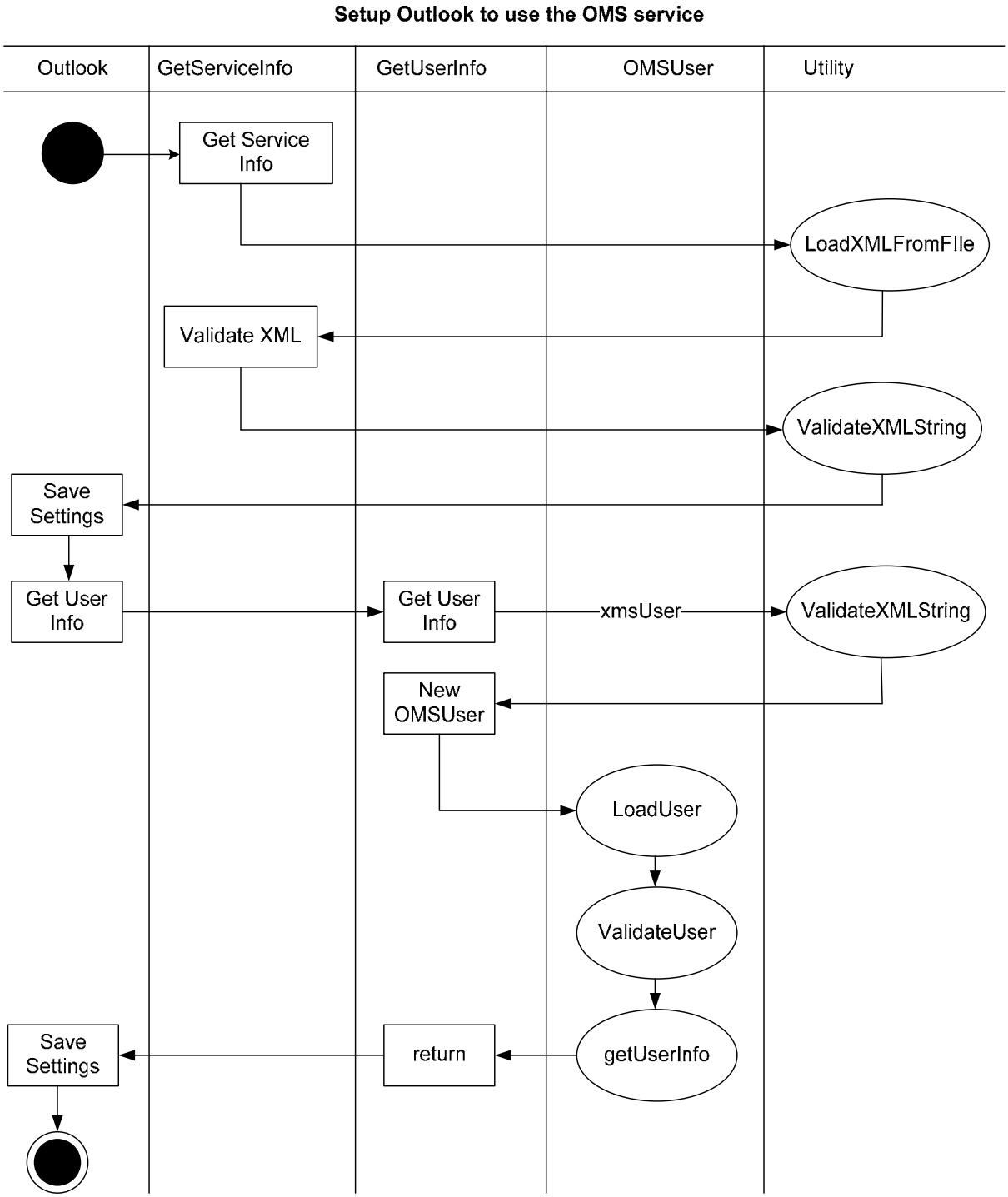
#	**Requirement Description**	**Comments**
UR3.01	The system will provide some general information, which can be viewed by unregistered users.	
UR3.02	User will be able to login to the system.	

#	**Requirement Description**	**Comments**
UR3.03	Users will be able to register for an account.	
UR3.04	Users will be able to define personal settings.	
UR3.05	Users will be able to view and accept terms and conditions.	

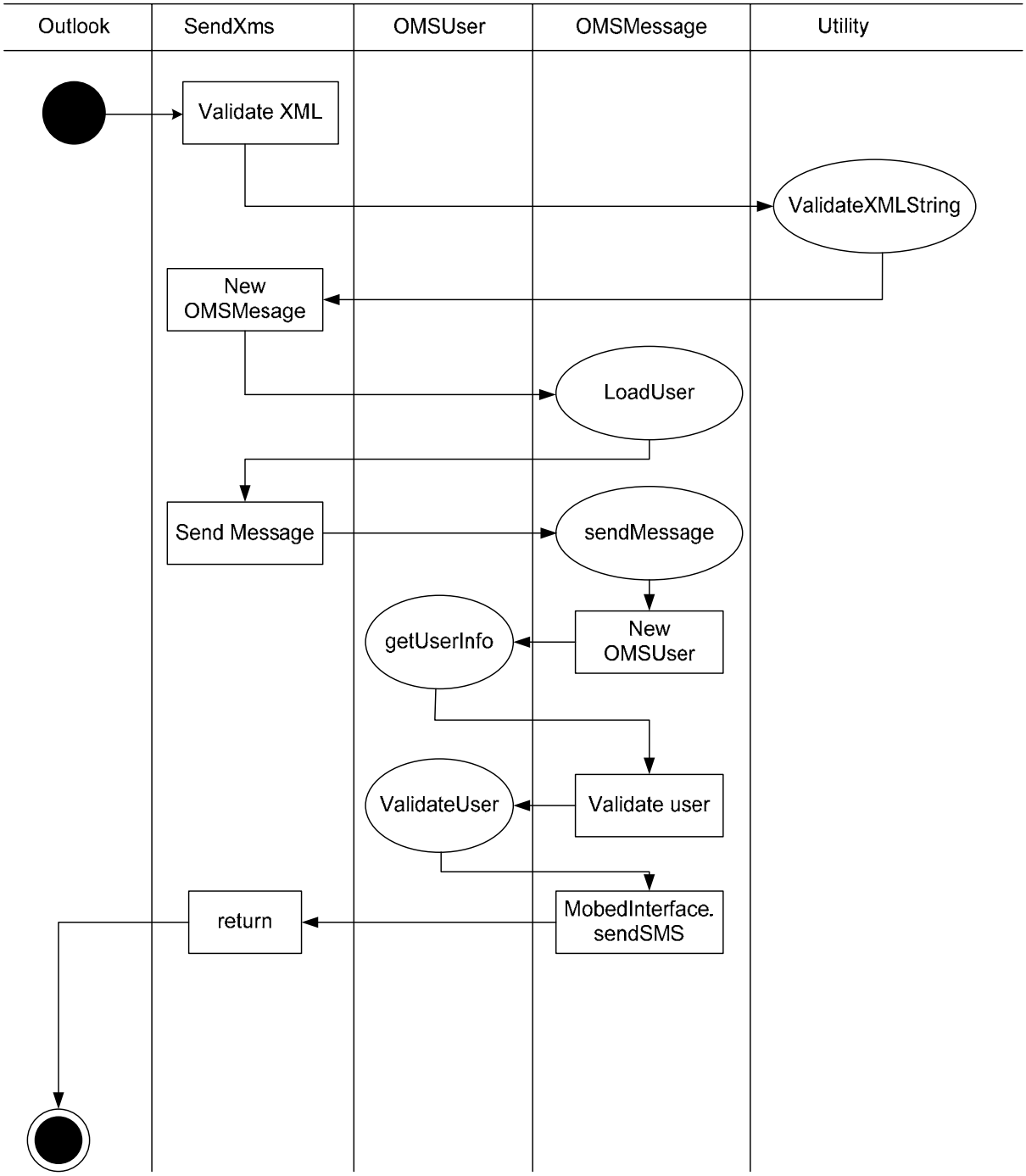
8.2 Appendix B – Entity Relationship Diagram



8.3 Appendix C - Class Diagram



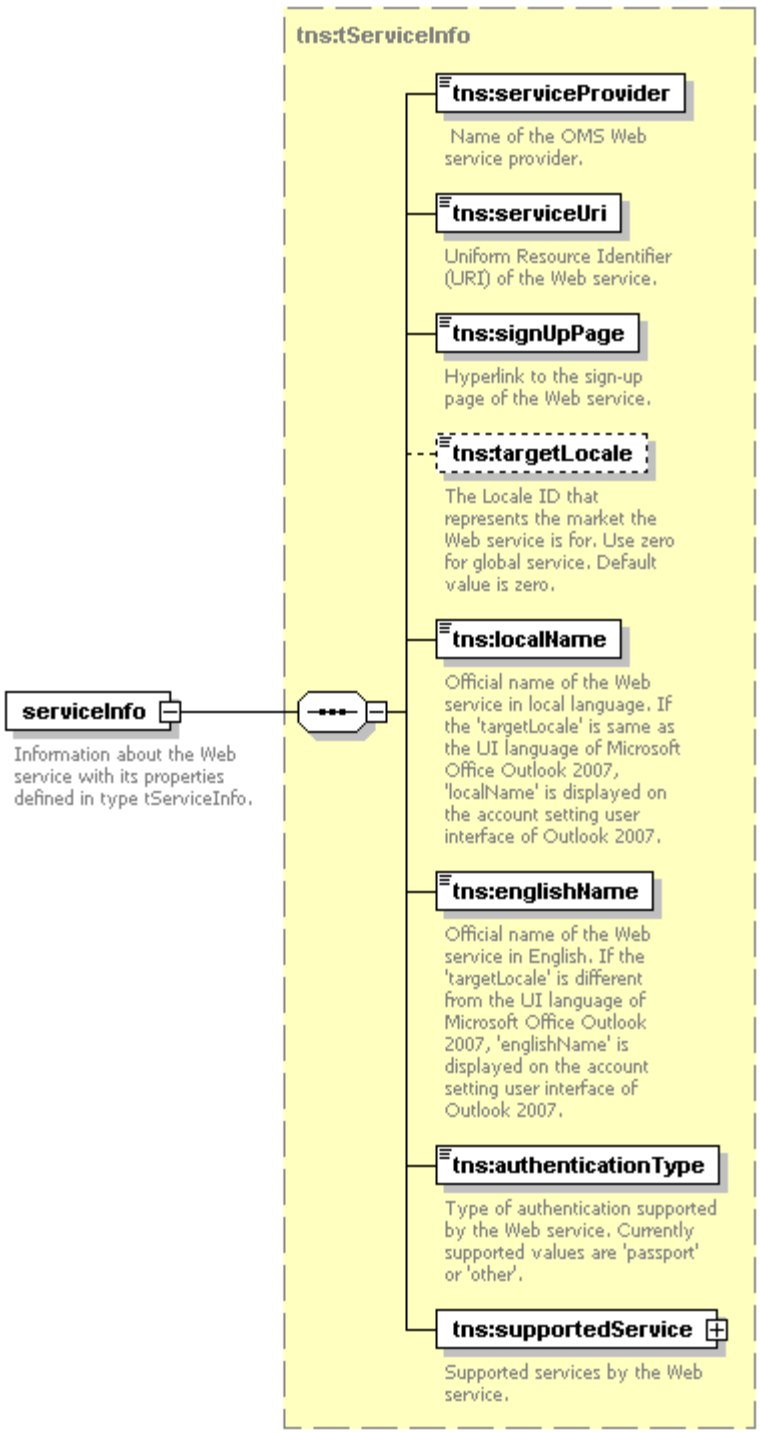
Send SMS Message



8.4 Appendix D – Schema Definitions

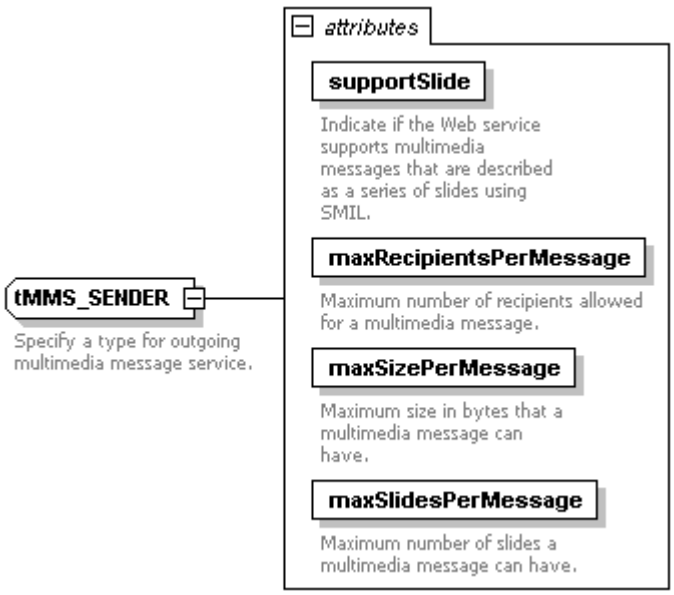
serviceInfo Schema – This schema defines the data structure for configuration parameters of the OMS service.

element **serviceInfo**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
type	**tns:tServiceInfo**

properties	content complex
children	**tns:serviceProvider tns:serviceUri tns:signUpPage tns:targetLocale tns:localName tns:englishName tns:authenticationType tns:supportedService**
annotation	documentation Information about the Web service with its properties defined in type tServiceInfo.
source	<xs:element name="serviceInfo" type="tns:tServiceInfo"> <xs:annotation> <xs:documentation>Information about the Web service with its properties defined in type tServiceInfo.</xs:documentation> </xs:annotation> </xs:element>

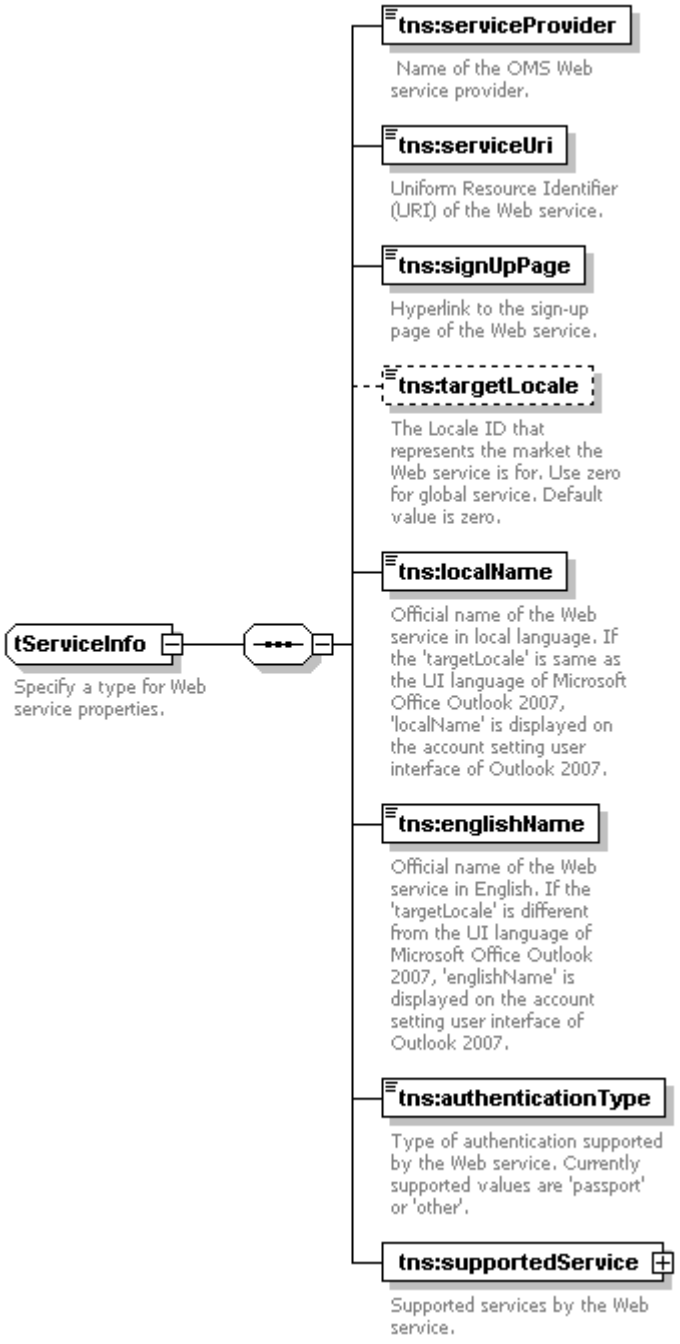
complexType **tMMS_SENDER**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS

used by	element **tSupportedService/MMS_SENDER**					
attributes	Name	Type	Use	Default	Fixed	annotation
	supportSlide	**xs:boolean**	required			documentation Indicate if the Web service supports multimedia messages that are described as a series of slides using SMIL.
	maxRecipientsPerMessage	**xs:unsignedInt**	required			documentation Maximum number of recipients allowed for a multimedia message.
	maxSizePerMessage	**xs:unsignedInt**	required			documentation Maximum size in bytes that a multimedia message can have.
	maxSlidesPerMessage	**xs:unsignedInt**	required			documentation Maximum number of slides a

	multimedia message can have.
annotation	documentation Specify a type for outgoing multimedia message service.
source	<xs:complexType name="tMMS_SENDER"> <xs:annotation> <xs:documentation>Specify a type for outgoing multimedia message service.</xs:documentation> </xs:annotation> <xs:attribute name="supportSlide" type="xs:boolean" use="required"> <xs:annotation> <xs:documentation>Indicate if the Web service supports multimedia messages that are described as a series of slides using SMIL. </xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="maxRecipientsPerMessage" type="xs:unsignedInt" use="required"> <xs:annotation> <xs:documentation>Maximum number of recipients allowed for a multimedia message.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="maxSizePerMessage" type="xs:unsignedInt" use="required"> <xs:annotation> <xs:documentation>Maximum size in bytes that a multimedia message can have.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="maxSlidesPerMessage" type="xs:unsignedInt" use="required"> <xs:annotation> <xs:documentation>Maximum number of slides a multimedia message can have.</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>

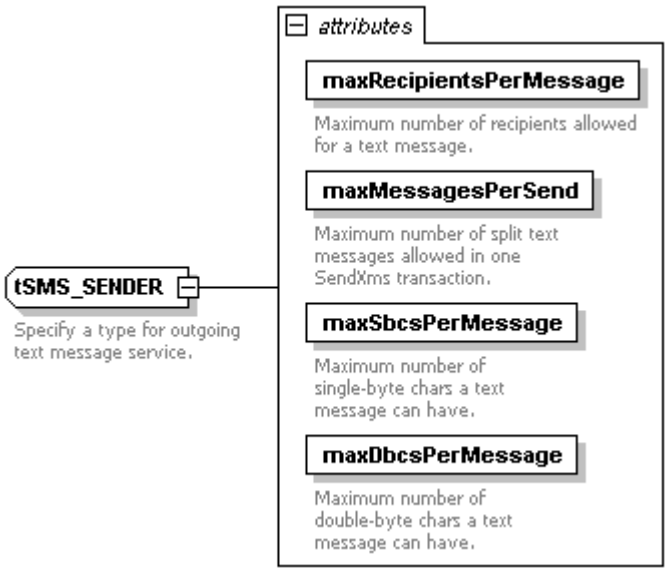
complexType **tServiceInfo**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS

children	**tns:serviceProvider** **tns:serviceUri** **tns:signUpPage** **tns:targetLocale** **tns:localName** **tns:englishName** **tns:authenticationType** **tns:supportedService**
used by	element **serviceInfo**
annotation	documentation Specify a type for Web service properties.
source	<xs:complexType name="tServiceInfo"> <xs:annotation> <xs:documentation>Specify a type for Web service properties.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="serviceProvider" type="xs:string"> <xs:annotation> <xs:documentation> Name of the OMS Web service provider.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="serviceUri" type="xs:string"> <xs:annotation> <xs:documentation>Uniform Resource Identifier (URI) of the Web service.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="signUpPage" type="xs:string"> <xs:annotation> <xs:documentation>Hyperlink to the sign-up page of the Web service.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="targetLocale" type="xs:unsignedShort" minOccurs="0"> <xs:annotation> <xs:documentation>The Locale ID that represents the market the Web service is for. Use zero for global service. Default value is zero. </xs:documentation> </xs:annotation> </xs:element> <xs:element name="localName" type="xs:string">

```
      <xs:annotation>
        <xs:documentation>Official name of the Web service in local language. If the 'targetLocale' is same as the UI language of Microsoft Office Outlook 2007, 'localName' is displayed on the account setting user interface of Outlook 2007.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="englishName" type="xs:string">
      <xs:annotation>
        <xs:documentation>Official name of the Web service in English. If the 'targetLocale' is different from the UI language of Microsoft Office Outlook 2007, 'englishName' is displayed on the account setting user interface of Outlook 2007.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="authenticationType" type="xs:string">
      <xs:annotation>
        <xs:documentation>Type of authentication supported by the Web service. Currently supported values are 'passport' or 'other'.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="supportedService" type="tns:tSupportedService">
      <xs:annotation>
        <xs:documentation>Supported services by the Web service.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

complexType **tSMS_SENDER**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
used by	element **tSupportedService/SMS_SENDER**

attributes

Name	Type	Use	Default	Fixed	annotation
maxRecipientsPerMessage	**xs:unsignedInt**	required			documentation Maximum number of recipients allowed for a text message.
maxMessagesPerSend	**xs:unsignedInt**	required			documentation Maximum number of split text messages allowed in one SendXms

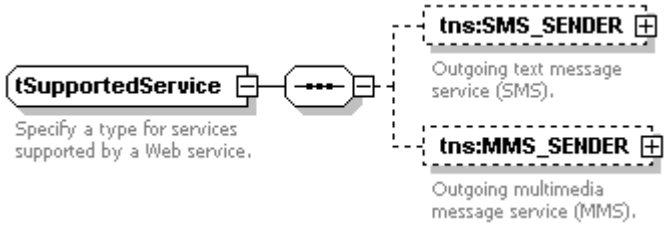
				transaction.
	maxSbcsPerMessage	**xs:unsignedInt**	required	documentation Maximum number of single-byte chars a text message can have.
	maxDbcsPerMessage	**xs:unsignedInt**	required	documentation Maximum number of double-byte chars a text message can have.
annotation	documentation Specify a type for outgoing text message service.			

source

```
<xs:complexType name="tSMS_SENDER">
 <xs:annotation>
  <xs:documentation>Specify a type for outgoing text message service.</xs:documentation>
 </xs:annotation>
 <xs:attribute name="maxRecipientsPerMessage" type="xs:unsignedInt" use="required">
  <xs:annotation>
   <xs:documentation>Maximum number of recipients allowed for a text message.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="maxMessagesPerSend" type="xs:unsignedInt" use="required">
  <xs:annotation>
   <xs:documentation>Maximum number of split text messages allowed in one SendXms
transaction.</xs:documentation>
```

	</xs:annotation> </xs:attribute> <xs:attribute name="maxSbcsPerMessage" type="xs:unsignedInt" use="required"> <xs:annotation> <xs:documentation>Maximum number of single-byte chars a text message can have.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="maxDbcsPerMessage" type="xs:unsignedInt" use="required"> <xs:annotation> <xs:documentation>Maximum number of double-byte chars a text message can have.</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>

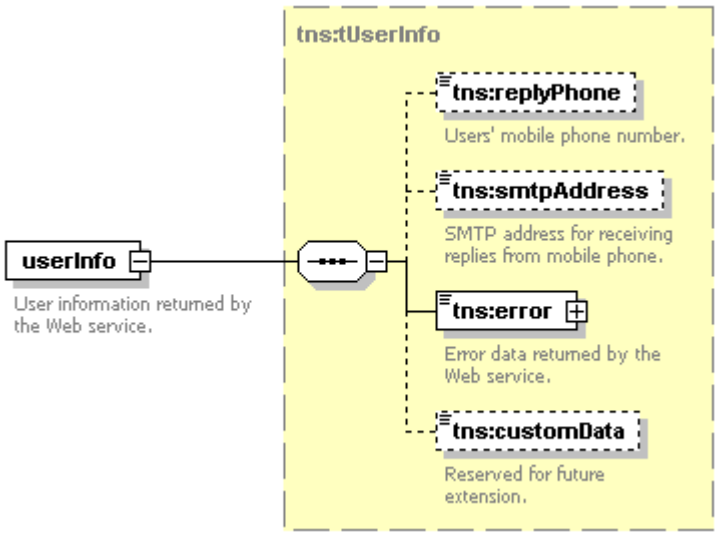
complexType **tSupportedService**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:SMS_SENDER** **tns:MMS_SENDER**
used by	element **tServiceInfo/supportedService**
annotation	documentation Specify a type for services supported by a Web service.
source	<xs:complexType name="tSupportedService">

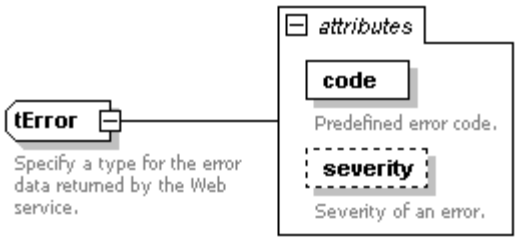
```
<xs:annotation>
  <xs:documentation>Specify a type for services supported by a Web service.</xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="SMS_SENDER" type="tns:tSMS_SENDER" minOccurs="0">
    <xs:annotation>
      <xs:documentation>Outgoing text message service (SMS).</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="MMS_SENDER" type="tns:tMMS_SENDER" minOccurs="0">
    <xs:annotation>
      <xs:documentation>Outgoing multimedia message service (MMS).</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
</xs:complexType>
```

userInfo Schema – This schema defines the data structure for user configuration parameters

element **userInfo**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
type	**tns:tUserInfo**
properties	content complex
children	**tns:replyPhone** **tns:smtpAddress** **tns:error** **tns:customData**
annotation	documentation User information returned by the Web service.
source	<xs:element name="userInfo" type="tns:tUserInfo"> <xs:annotation> <xs:documentation>User information returned by the Web service.</xs:documentation> </xs:annotation> </xs:element>

complexType **tError**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
type	extension of **xs:string**
properties	base xs:string
used by	element **tUserInfo/error**

attributes

Name	Type	Use	Default	Fixed	annotation
code	**xs:string**	required			documentation Predefined error code.
severity	**xs:string**	optional			documentation Severity of an error.

annotation

documentation

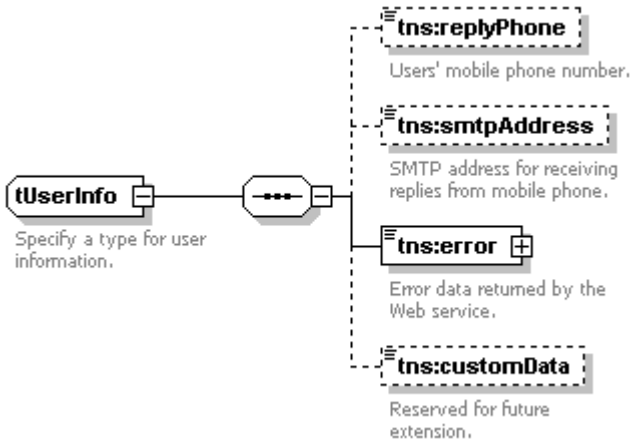
Specify a type for the error data returned by the Web service.

source

```
<xs:complexType name="tError">
 <xs:annotation>
  <xs:documentation>Specify a type for the error data returned by the Web service.</xs:documentation>
 </xs:annotation>
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="code" type="xs:string" use="required">
```

```
      <xs:annotation>
        <xs:documentation>Predefined error code.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="severity" type="xs:string" use="optional">
      <xs:annotation>
        <xs:documentation>Severity of an error.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

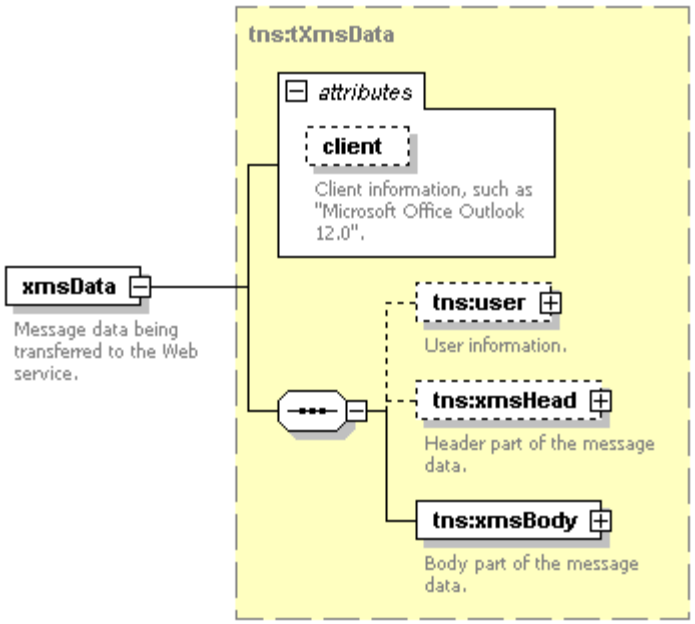
complexType **tUserInfo**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:replyPhone** **tns:smtpAddress** **tns:error** **tns:customData**
used by	element **userInfo**
annotation	documentation

	Specify a type for user information.
source	<xs:complexType name="tUserInfo"> <xs:annotation> <xs:documentation>Specify a type for user information.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="replyPhone" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Users' mobile phone number.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="smtpAddress" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>SMTP address for receiving replies from mobile phone.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="error" type="tns:tError"> <xs:annotation> <xs:documentation>Error data returned by the Web service.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="customData" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Reserved for future extension.</xs:documentation> </xs:annotation> </xs:element> </xs:sequence> </xs:complexType>

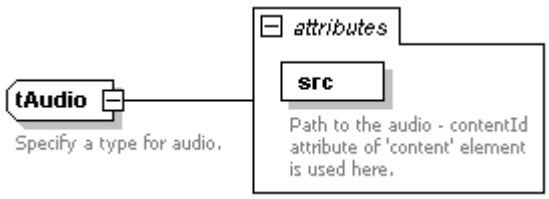
xmsData Schema – This schema defines the data structure for the SMS message passed from the Outlook client to the OMS web service.

element **xmsData**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
type	**tns:tXmsData**
properties	content complex
children	**tns:user** **tns:xmsHead** **tns:xmsBody**
attributes	Name: client Type: **xs:string** Use: Default: Fixed: annotation: documentation Client information, such as "Microsoft Office Outlook 12.0".
annotation	documentation

	Message data being transferred to the Web service.
source	<xs:element name="xmsData" type="tns:tXmsData"> <xs:annotation> <xs:documentation>Message data being transferred to the Web service.</xs:documentation> </xs:annotation> </xs:element>

complexType **tAudio**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
used by	element **tPar/audio**

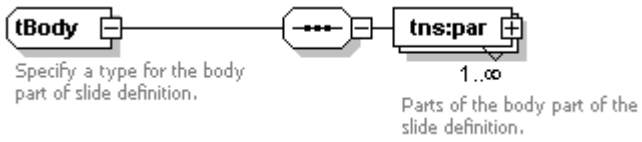
attributes

Name	Type	Use	Default	Fixed	annotation
src	**xs:string**	required			documentation Path to the audio - contentId attribute of 'content' element is used here.

annotation	documentation

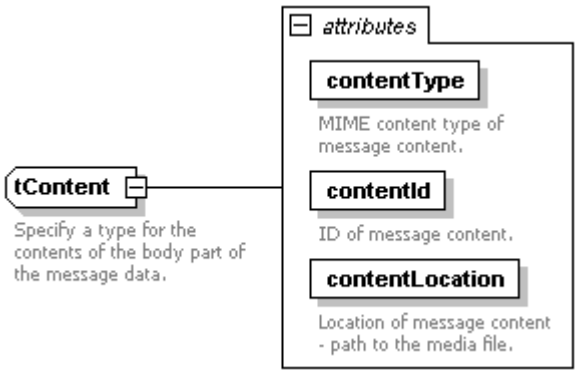
	Specify a type for audio.
source	<xs:complexType name="tAudio"> <xs:annotation> <xs:documentation>Specify a type for audio.</xs:documentation> </xs:annotation> <xs:attribute name="src" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Path to the audio - contentId attribute of 'content' element is used here.</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>

complexType **tBody**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:par**
used by	element **tMmsSlides/body**
annotation	documentation Specify a type for the body part of slide definition.
source	<xs:complexType name="tBody"> <xs:annotation> <xs:documentation>Specify a type for the body part of slide definition.</xs:documentation>

```
    </xs:annotation>
    <xs:sequence>
      <xs:element name="par" type="tns:tPar" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Parts of the body part of the slide definition.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

complexType **tContent**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
type	extension of **xs:string**
properties	base xs:string
used by	element **tXmsBody/content**

attributes	Name	Type	Use	Default	Fixed	annotation
	contentType	**xs:string**	required			documentation MIME content

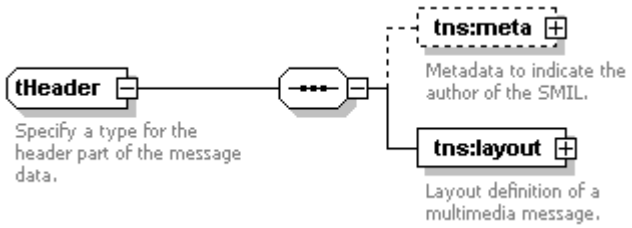
				type of message content.
	contentId	**xs:string**	required	documentation ID of message content.
	contentLocation	**xs:string**	required	documentation Location of message content - path to the media file.
annotation	documentation Specify a type for the contents of the body part of the message data.			

source

```
<xs:complexType name="tContent">
 <xs:annotation>
  <xs:documentation>Specify a type for the contents of the body part of the message data.</xs:documentation>
 </xs:annotation>
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="contentType" type="xs:string" use="required">
    <xs:annotation>
     <xs:documentation>MIME content type of message content.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="contentId" type="xs:string" use="required">
    <xs:annotation>
     <xs:documentation>ID of message content.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
```

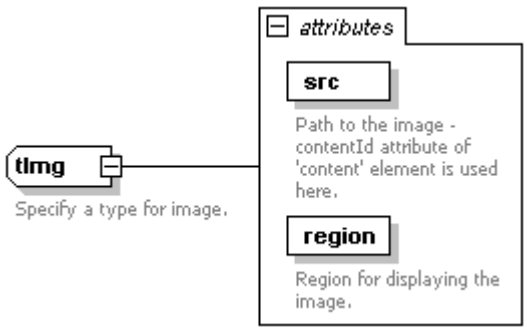
	<xs:attribute name="contentLocation" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Location of message content - path to the media file.</xs:documentation> </xs:annotation> </xs:attribute> </xs:extension> </xs:simpleContent> </xs:complexType>

complexType **tHeader**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:meta** **tns:layout**
used by	element **tMmsSlides/head**
annotation	documentation Specify a type for the header part of the message data.
source	<xs:complexType name="tHeader"> <xs:annotation> <xs:documentation>Specify a type for the header part of the message data.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="meta" type="tns:tMeta" minOccurs="0">

```
      <xs:annotation>
        <xs:documentation>Metadata to indicate the author of the SMIL.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="layout" type="tns:tLayout">
      <xs:annotation>
        <xs:documentation>Layout definition of a multimedia message.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

complexType **tImg**

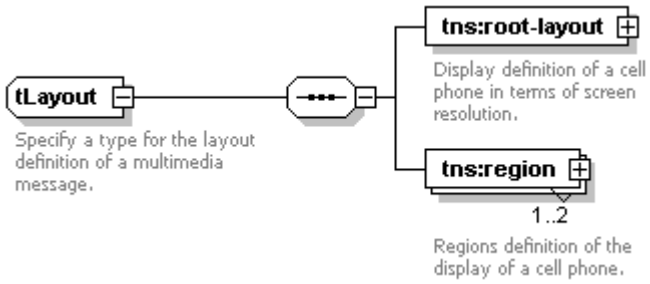
diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
used by	element **tPar/img**

attributes

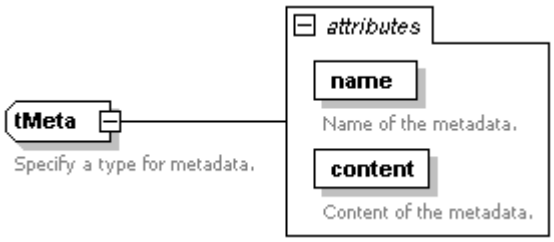
Name	Type	Use	Default	Fixed	annotation
src	**xs:string**	required			documentation Path to the image - contentId

	region **xs:string** required	attribute of 'content' element is used here. documentation Region for displaying the image.
annotation	documentation Specify a type for image.	
source	<xs:complexType name="tImg"> <xs:annotation> <xs:documentation>Specify a type for image.</xs:documentation> </xs:annotation> <xs:attribute name="src" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Path to the image - contentId attribute of 'content' element is used here.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="region" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Region for displaying the image.</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>	

complexType **tLayout**

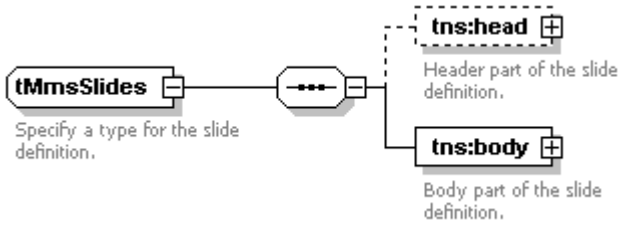
diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:root-layout** **tns:region**
used by	element **tHeader/layout**
annotation	documentation Specify a type for the layout definition of a multimedia message.
source	<xs:complexType name="tLayout"> <xs:annotation> <xs:documentation>Specify a type for the layout definition of a multimedia message.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="root-layout" type="tns:tRoot-layout"> <xs:annotation> <xs:documentation>Display definition of a cell phone in terms of screen resolution.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="region" type="tns:tRegion" maxOccurs="2"> <xs:annotation> <xs:documentation>Regions definition of the display of a cell phone.</xs:documentation> </xs:annotation> </xs:element> </xs:sequence> </xs:complexType>

complexType **tMeta**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
used by	element **tHeader/meta**
attributes	Name / Type / Use / Default / Fixed / annotation name — **xs:string** — required — documentation: Name of the metadata. content — **xs:string** — required — documentation: Content of the metadata.
annotation	documentation Specify a type for metadata.
source	<xs:complexType name="tMeta"> <xs:annotation> <xs:documentation>Specify a type for metadata.</xs:documentation> </xs:annotation> <xs:attribute name="name" type="xs:string" use="required"> <xs:annotation>

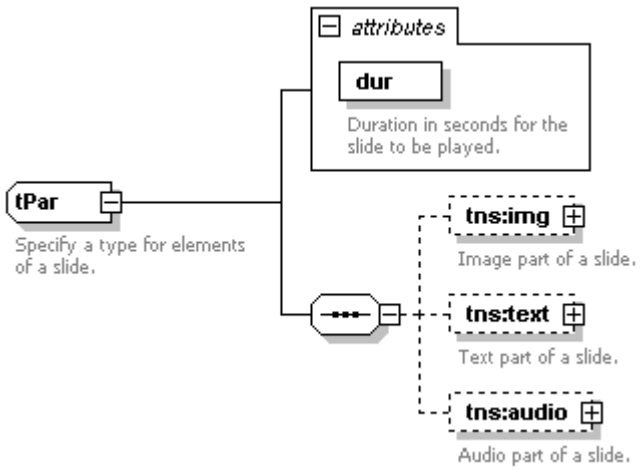
	<xs:documentation>Name of the metadata.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="content" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Content of the metadata.</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>

complexType **tMmsSlides**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:head** **tns:body**
used by	element **tXmsBody/mmsSlides**
annotation	documentation Specify a type for the slide definition.
source	<xs:complexType name="tMmsSlides"> <xs:annotation> <xs:documentation>Specify a type for the slide definition.</xs:documentation> </xs:annotation> <xs:sequence>

```
<xs:element name="head" type="tns:tHeader" minOccurs="0">
 <xs:annotation>
  <xs:documentation>Header part of the slide definition.</xs:documentation>
 </xs:annotation>
</xs:element>
<xs:element name="body" type="tns:tBody">
 <xs:annotation>
  <xs:documentation>Body part of the slide definition.</xs:documentation>
 </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
```

complexType **tPar**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:img tns:text tns:audio**
used by	element **tBody/par**

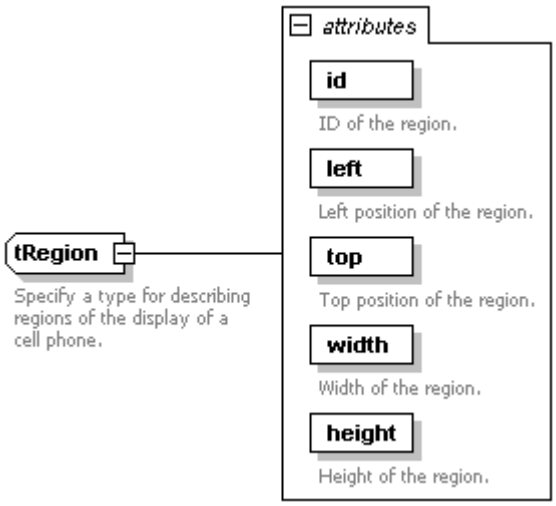
attributes	Name	Type	Use	Default	Fixed	annotation
	dur	**xs:unsignedInt**	required			documentation Duration in seconds for the slide to be played.
annotation	documentation Specify a type for elements of a slide.					

source

```
<xs:complexType name="tPar">
 <xs:annotation>
  <xs:documentation>Specify a type for elements of a slide.</xs:documentation>
 </xs:annotation>
 <xs:sequence>
  <xs:element name="img" type="tns:tImg" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Image part of a slide.</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="text" type="tns:tText" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Text part of a slide.</xs:documentation>
   </xs:annotation>
  </xs:element>
  <xs:element name="audio" type="tns:tAudio" minOccurs="0">
   <xs:annotation>
    <xs:documentation>Audio part of a slide.</xs:documentation>
   </xs:annotation>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="dur" type="xs:unsignedInt" use="required">
  <xs:annotation>
```

	<xs:documentation>Duration in seconds for the slide to be played.</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>

complexType **tRegion**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
used by	element **tLayout/region**

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xs:string**	required			documentation ID of the region.
left	**xs:unsignedInt**	required			documentation Left position of the region.

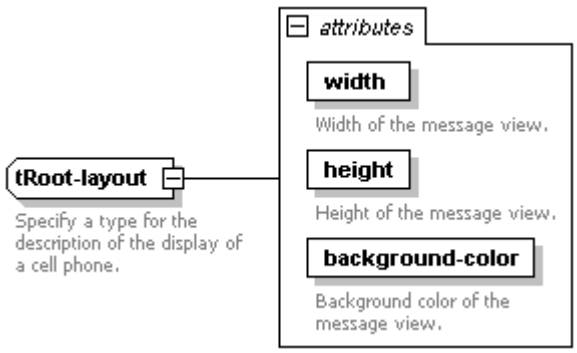
	top	**xs:unsignedInt**	required	documentation Top position of the region.
	width	**xs:unsignedInt**	required	documentation Width of the region.
	height	**xs:unsignedInt**	required	documentation Height of the region.
annotation	documentation Specify a type for describing regions of the display of a cell phone.			

source

```
<xs:complexType name="tRegion">
 <xs:annotation>
  <xs:documentation>Specify a type for describing regions of the display of a cell phone.</xs:documentation>
 </xs:annotation>
 <xs:attribute name="id" type="xs:string" use="required">
  <xs:annotation>
   <xs:documentation>ID of the region.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="left" type="xs:unsignedInt" use="required">
  <xs:annotation>
   <xs:documentation>Left position of the region.</xs:documentation>
  </xs:annotation>
 </xs:attribute>
 <xs:attribute name="top" type="xs:unsignedInt" use="required">
  <xs:annotation>
   <xs:documentation>Top position of the region.</xs:documentation>
  </xs:annotation>
```

```
    </xs:attribute>
    <xs:attribute name="width" type="xs:unsignedInt" use="required">
      <xs:annotation>
        <xs:documentation>Width of the region.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="height" type="xs:unsignedInt" use="required">
      <xs:annotation>
        <xs:documentation>Height of the region.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
```

complexType **tRoot-layout**

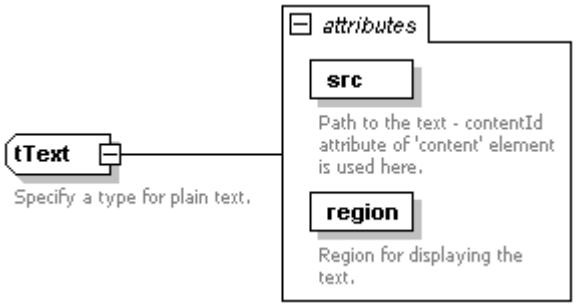
diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
used by	element **tLayout/root-layout**

attributes	Name	Type	Use	Default	Fixed	annotation
	width	**xs:unsignedInt**	required			documentation Width of the

				message view.
	height	**xs:unsignedByte**	required	documentation Height of the message view.
	background-color	**xs:string**	required	documentation Background color of the message view.
annotation	documentation Specify a type for the description of the display of a cell phone.			
source	<xs:complexType name="tRoot-layout"> <xs:annotation> <xs:documentation>Specify a type for the description of the display of a cell phone.</xs:documentation> </xs:annotation> <xs:attribute name="width" type="xs:unsignedInt" use="required"> <xs:annotation> <xs:documentation>Width of the message view.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="height" type="xs:unsignedByte" use="required"> <xs:annotation> <xs:documentation>Height of the message view.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="background-color" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Background color of the message view.</xs:documentation>			

```
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
```

complexType **tText**

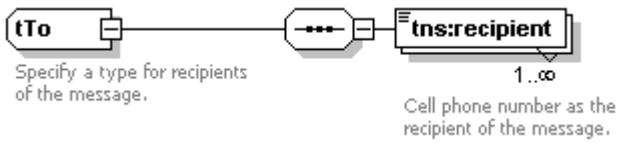
diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
used by	element **tPar/text**

attributes

Name	Type	Use	Default	Fixed	annotation
src	**xs:string**	required			documentation Path to the text - contentId attribute of 'content' element is used here.
region	**xs:string**	required			documentation Region for displaying the

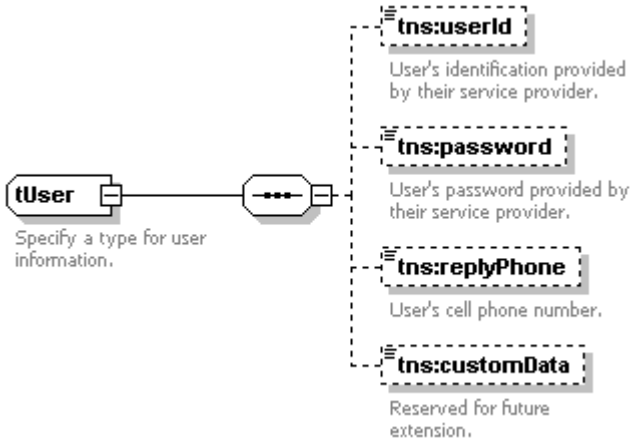
	text.
annotation	documentation Specify a type for plain text.
source	<xs:complexType name="tText"> <xs:annotation> <xs:documentation>Specify a type for plain text.</xs:documentation> </xs:annotation> <xs:attribute name="src" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Path to the text - contentId attribute of 'content' element is used here.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="region" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Region for displaying the text.</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>

complexType **tTo**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:recipient**

used by	element **tXmsHeader/to**
annotation	documentation Specify a type for recipients of the message.
source	<xs:complexType name="tTo"> <xs:annotation> <xs:documentation>Specify a type for recipients of the message.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="recipient" type="xs:string" maxOccurs="unbounded"> <xs:annotation> <xs:documentation>Cell phone number as the recipient of the message.</xs:documentation> </xs:annotation> </xs:element> </xs:sequence> </xs:complexType>

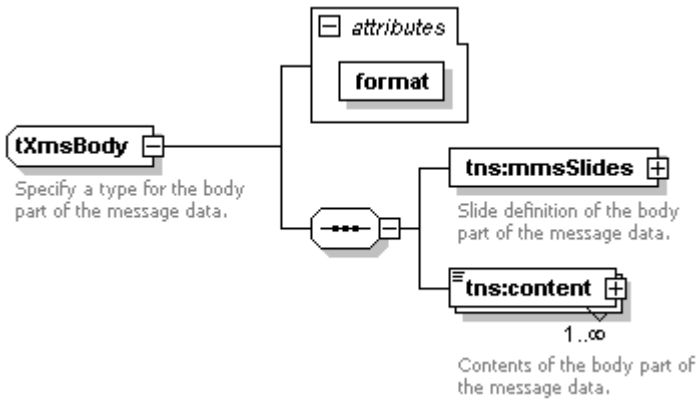
complexType **tUser**

diagram	

namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:userId** **tns:password** **tns:replyPhone** **tns:customData**
used by	element **tXmsData/user**
annotation	documentation Specify a type for user information.
source	<xs:complexType name="tUser"> <xs:annotation> <xs:documentation>Specify a type for user information.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="userId" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>User's identification provided by their service provider.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="password" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>User's password provided by their service provider.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="replyPhone" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>User's cell phone number.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="customData" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Reserved for future extension.</xs:documentation> </xs:annotation> </xs:element> </xs:sequence>

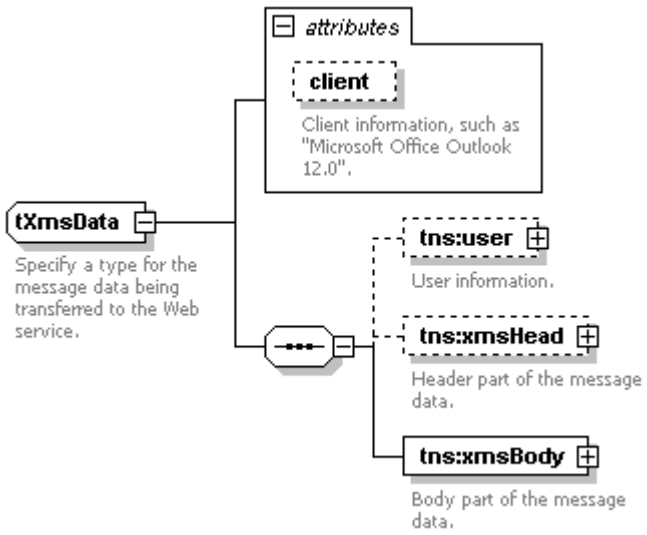
	</xs:complexType>

complexType **tXmsBody**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:mmsSlides** **tns:content**
used by	element **tXmsData/xmsBody**
attributes	Name: format; Type: **xs:string**; Use: required; Default: ; Fixed: ; annotation:
annotation	documentation Specify a type for the body part of the message data.
source	<xs:complexType name="tXmsBody"> <xs:annotation> <xs:documentation>Specify a type for the body part of the message data.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="mmsSlides" type="tns:tMmsSlides">

```
      <xs:annotation>
        <xs:documentation>Slide definition of the body part of the message data.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="content" type="tns:tContent" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Contents of the body part of the message data.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="format" type="xs:string" use="required"/>
</xs:complexType>
```

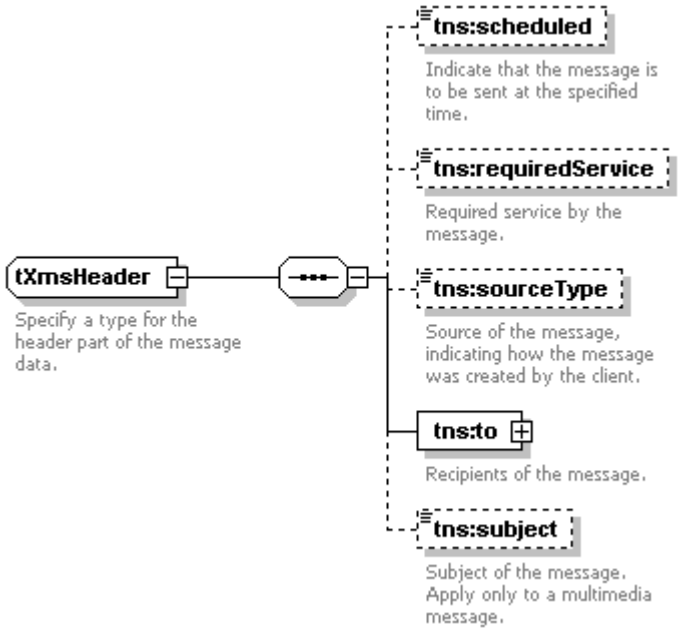
complexType **tXmsData**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:user tns:xmsHead tns:xmsBody**

used by	element **xmsData**
attributes	Name: client Type: xs:string Use: Default: Fixed: annotation: documentation Client information, such as "Microsoft Office Outlook 12.0".
annotation	documentation Specify a type for the message data being transferred to the Web service.
source	<xs:complexType name="tXmsData"> <xs:annotation> <xs:documentation>Specify a type for the message data being transferred to the Web service.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="user" type="tns:tUser" minOccurs="0"> <xs:annotation> <xs:documentation>User information.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="xmsHead" type="tns:tXmsHeader" minOccurs="0"> <xs:annotation> <xs:documentation>Header part of the message data.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="xmsBody" type="tns:tXmsBody"> <xs:annotation> <xs:documentation>Body part of the message data.</xs:documentation>

```
    </xs:annotation>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="client" type="xs:string">
  <xs:annotation>
   <xs:documentation>Client information, such as "Microsoft Office Outlook 12.0".</xs:documentation>
  </xs:annotation>
 </xs:attribute>
</xs:complexType>
```

complexType **tXmsHeader**

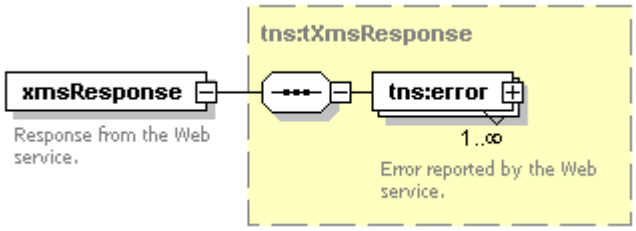
diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:scheduled** **tns:requiredService** **tns:sourceType** **tns:to** **tns:subject**
used by	element **tXmsData/xmsHead**

annotation	documentation Specify a type for the header part of the message data.
source	<xs:complexType name="tXmsHeader"> <xs:annotation> <xs:documentation>Specify a type for the header part of the message data.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="scheduled" type="xs:dateTime" minOccurs="0"> <xs:annotation> <xs:documentation>Indicate that the message is to be sent at the specified time.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="requiredService" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Required service by the message.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="sourceType" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Source of the message, indicating how the message was created by the client. </xs:documentation> </xs:annotation> </xs:element> <xs:element name="to" type="tns:tTo"> <xs:annotation> <xs:documentation>Recipients of the message.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="subject" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Subject of the message. Apply only to a multimedia message.</xs:documentation> </xs:annotation> </xs:element>

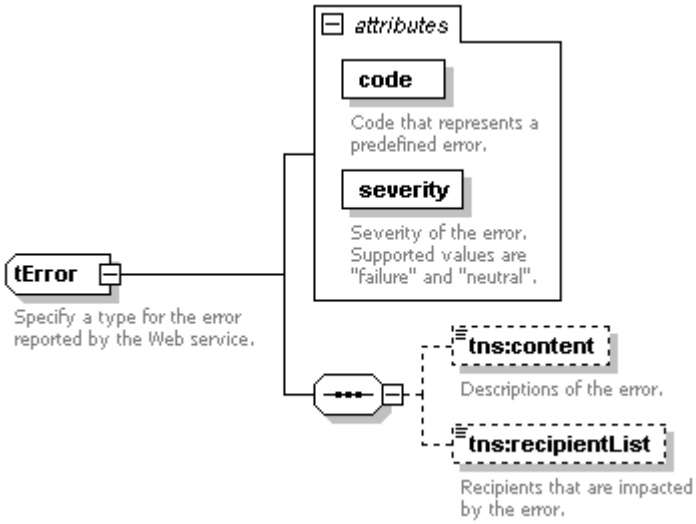
	</xs:sequence> </xs:complexType>

xmsResponse Schema – This schema defines the data structure for the response from the OMS web service to the Outlook client after the SMS message has been process and sent.

element **xmsResponse**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
type	**tns:tXmsResponse**
properties	content complex
children	**tns:error**
annotation	documentation Response from the Web service.
source	<xs:element name="xmsResponse" type="tns:tXmsResponse"> <xs:annotation> <xs:documentation>Response from the Web service.</xs:documentation> </xs:annotation> </xs:element>

complexType **tError**

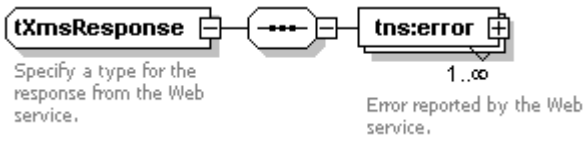
diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:content** **tns:recipientList**
used by	element **tXmsResponse/error**

attributes

Name	Type	Use	Default	Fixed	annotation
code	**xs:string**	required			documentation Code that represents a predefined error.
severity	**xs:string**	required			documentation Severity of the error. Supported values are "failure" and

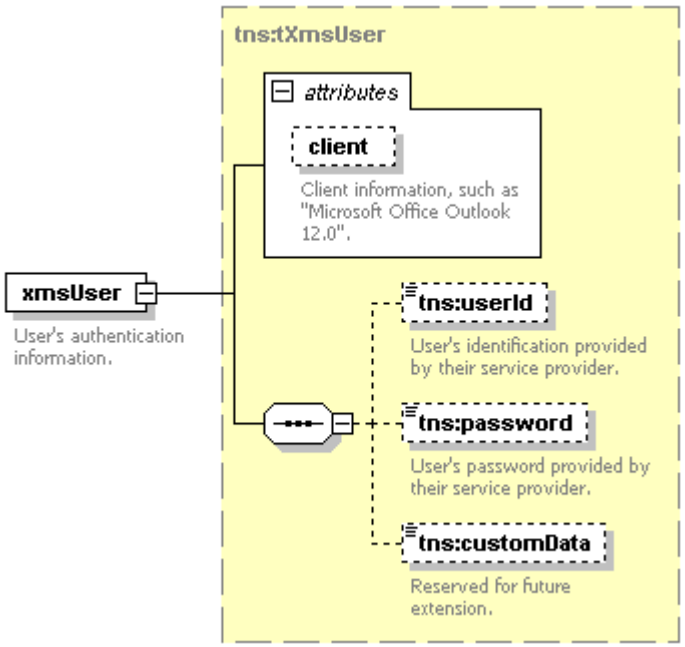
	"neutral".
annotation	documentation Specify a type for the error reported by the Web service.
source	<xs:complexType name="tError"> <xs:annotation> <xs:documentation>Specify a type for the error reported by the Web service.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="content" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Descriptions of the error.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="recipientList" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Recipients that are impacted by the error.</xs:documentation> </xs:annotation> </xs:element> </xs:sequence> <xs:attribute name="code" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Code that represents a predefined error.</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="severity" type="xs:string" use="required"> <xs:annotation> <xs:documentation>Severity of the error. Supported values are "failure" and "neutral".</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>

complexType **tXmsResponse**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:error**
used by	element **xmsResponse**
annotation	documentation Specify a type for the response from the Web service.
source	<xs:complexType name="tXmsResponse"> <xs:annotation> <xs:documentation>Specify a type for the response from the Web service.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="error" type="tns:tError" maxOccurs="unbounded"> <xs:annotation> <xs:documentation>Error reported by the Web service.</xs:documentation> </xs:annotation> </xs:element> </xs:sequence> </xs:complexType>

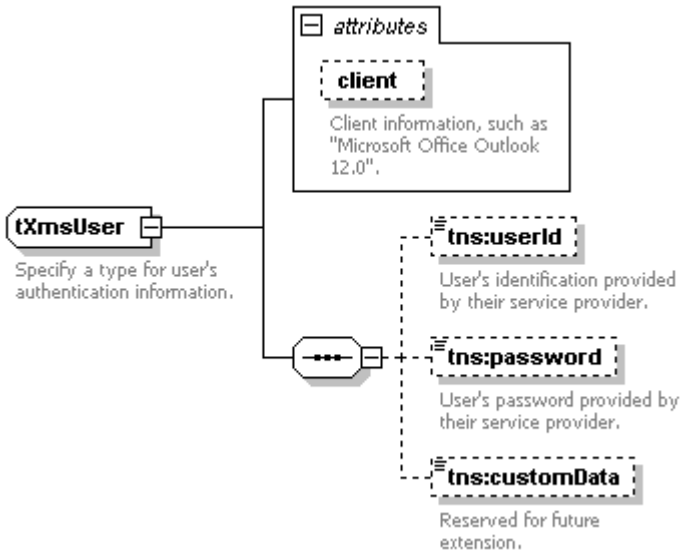
xmsUser – This schema defines the data structure for user information sent from the Outlook client to the OMS web service. This information is used to retrieve additional user information and return it in userInfo format.

element **xmsUser**

<table>
<tr><td>diagram</td><td> 
</td></tr>
<tr><td>namespace</td><td>http://schemas.microsoft.com/office/Outlook/2006/OMS</td></tr>
<tr><td>type</td><td>tns:tXmsUser</td></tr>
<tr><td>properties</td><td>content complex</td></tr>
<tr><td>children</td><td>tns:userId tns:password tns:customData</td></tr>
<tr><td>attributes</td><td>Name Type Use Default Fixed annotation
client xs:string documentation
Client information, such as "Microsoft</td></tr>
</table>

	Office Outlook 12.0".
annotation	documentation User's authentication information.
source	<xs:element name="xmsUser" type="tns:tXmsUser"> <xs:annotation> <xs:documentation>User's authentication information.</xs:documentation> </xs:annotation> </xs:element>

complexType **tXmsUser**

diagram	
namespace	http://schemas.microsoft.com/office/Outlook/2006/OMS
children	**tns:userId** **tns:password** **tns:customData**
used by	element **xmsUser**

attributes	Name	Type	Use	Default	Fixed	annotation
	client	**xs:string**				documentation Client information, such as "Microsoft Office Outlook 12.0".
annotation	documentation Specify a type for user's authentication information.					
source	<xs:complexType name="tXmsUser"> <xs:annotation> <xs:documentation>Specify a type for user's authentication information.</xs:documentation> </xs:annotation> <xs:sequence> <xs:element name="userId" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>User's identification provided by their service provider.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="password" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>User's password provided by their service provider.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="customData" type="xs:string" minOccurs="0"> <xs:annotation> <xs:documentation>Reserved for future extension.</xs:documentation> </xs:annotation> </xs:element> </xs:sequence>					

	<xs:attribute name="client" type="xs:string"> <xs:annotation> <xs:documentation>Client information, such as "Microsoft Office Outlook 12.0".</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType>

8.5 Appendix E – Source Code

OMS Lib Project

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings"
type="System.Configuration.ApplicationSettingsGroup, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
            <section name="oms_lib.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
        </sectionGroup>
    </configSections>
    <applicationSettings>
        <oms_lib.Properties.Settings>
            <setting name="oms_lib_mobed_SendSMS" serializeAs="String">

<value>http://75.126.204.15:80/axis2/services/SendSMS</value>
            </setting>
        </oms_lib.Properties.Settings>
    </applicationSettings>
</configuration>
```

MobedInterface.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Net;
using System.IO;

namespace oms_lib
{
    /*
     * This class is used to send SMS messages to MOBED for delivery.
     */
    public static class MobedInterface
    {

        /*
         * sendSMS takes a recipientList array, the number of messages
that are going to be sent and
         * messages split into 160 char chunks
         */
```

```
        public static Boolean sendSMS(String[] recipientList, int
numberOfMsg, String[,] message, int u_id)
        {
            mobed.sendSMS sms = null;
            mobed.sendSMSResponse s;
            mobed.SendSMS service = null;
            Boolean ret = false;
            String countryCode ="001";
            int carrierID = 0;
            try
            {
                for (int i = 0; i < recipientList.Length; i++)
                {
                    for (int z = 0; z < numberOfMsg; z++)
                    {

                        //Get the carrier id for the current recipient
                        carrierID =
getCarrierId(recipientList[i].ToString(),u_id);

                        //If the carrier ID was found send the message to
MOBED
                        if(carrierID != 0)
                        {
                            sms = new mobed.sendSMS();

                            sms.param0 = countryCode +
recipientList[i].ToString();
                            sms.param1 = "90947";
                            sms.param2 = message[z, 3].ToString();
                            sms.param3 = carrierID;
                            sms.param4 = "mo";
                            sms.param5 = "uncw";

                            service = new mobed.SendSMS();

                            s = service.sendSMS(sms);
                        }
                    }
                }
            }
            catch (Exception e)
            {
                Utility.logError(u_id, "sendSMS:" + e.Message);
                return true;
            }

            return ret;
        }

        /*
         * Attempts to find the recipient's carrier id in the database
         */
        private static int getCarrierId(String strPhone, int u_id)
        {
```

```
            SqlCommand cmd = null;
            int ret;

            try
            {
                cmd = new SqlCommand();
                cmd.Connection = Utility.getDBConnection();
                cmd.CommandType = System.Data.CommandType.StoredProcedure;
                cmd.CommandText = "getCarrierId";
                cmd.Parameters.Add("@phoneNumber",
System.Data.SqlDbType.VarChar).Value = strPhone;
                cmd.Parameters.Add("@u_id",
System.Data.SqlDbType.Int).Value = u_id;
                cmd.Parameters.Add("ret",
System.Data.SqlDbType.Int).Direction =
System.Data.ParameterDirection.ReturnValue;
                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                cmd.Connection.Close();

                if
(!cmd.Parameters["ret"].Value.Equals(System.DBNull.Value))
                    ret = (int)cmd.Parameters["ret"].Value;
                else
                {
                    //ret = 0;
                    ret = lookUpClientId(strPhone, u_id);
                    if (ret > 0)
                    {
                        cmd = new SqlCommand();
                        cmd.Connection = Utility.getDBConnection();
                        cmd.CommandType =
System.Data.CommandType.StoredProcedure;
                        cmd.CommandText = "saveCarrierId";
                        cmd.Parameters.Add("@phoneNumber",
System.Data.SqlDbType.VarChar).Value = strPhone;
                        cmd.Parameters.Add("@u_id",
System.Data.SqlDbType.Int).Value = u_id;
                        cmd.Parameters.Add("@carrierID",
System.Data.SqlDbType.Int).Value = ret;

                        cmd.Connection.Open();
                        cmd.ExecuteNonQuery();
                        cmd.Connection.Close();

                    }
                }
                return ret;
            }
            catch (Exception e)
            {
                Utility.logError(u_id, "getCarrierId:" + e.Message);
                throw;
            }
            finally
```

```
            {
                if (cmd != null)
                    cmd.Dispose();
            }


        }

        /*
         * Call the reverse lookup feature at whitepages.com to get the
carrier id.
         *
         */
        private static int lookUpClientId(String strPhone, int u_id)
        {
            WebRequest request = null;
            WebResponse response = null;
            StreamReader reader = null;
            String url, result;
            int ret=0;
            SqlCommand cmd = null;
            try
            {
                url =
"http://www.whitepages.com/search/ReversePhone?full_phone="+strPhone;
                request = WebRequest.Create(url);
                response = request.GetResponse();
                reader = new StreamReader(response.GetResponseStream());

                result = reader.ReadToEnd();

                //Find the string "Provider: "
                result = result.Substring(result.IndexOf("Provider: "));

                //Remove Everything after the provider name
                result = result.Substring(0, result.IndexOf("</span>"));

                //Remove Provider:
                result = result.Substring(result.IndexOf(":") + 1);

                //Store carrier code
                cmd = new SqlCommand();
                cmd.Connection = Utility.getDBConnection();
                cmd.CommandType = System.Data.CommandType.StoredProcedure;
                cmd.CommandText = "getCarrierIdByName";
                cmd.Parameters.Add("@providerName",
System.Data.SqlDbType.VarChar).Value = result;
                cmd.Parameters.Add("ret",
System.Data.SqlDbType.Int).Direction =
System.Data.ParameterDirection.ReturnValue;
                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                cmd.Connection.Close();
```

```
                if
(!cmd.Parameters["ret"].Value.Equals(System.DBNull.Value))
                    ret = (int)cmd.Parameters["ret"].Value;
                else
                {
                    ret = 0;
                }

            }
            catch (Exception e )
            {
                Utility.logError(u_id, "lookUpClientId:" + e.Message);
                throw;
            }
            return ret;

        }


    }
}
```

OMSMessage.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.Data.SqlClient;
using System.Data;
using System.IO;

namespace oms_lib
{
    /*
     * This class defines the datastructure for messages and provides
     * the processes needed to parse the message.
     */
    public class OMSMessage
    {
        String client, userID, password, replyPhone, customData;
        String requiredService, sourceType;
        String[] recipientList;
        String msgContent;
        String[,] message;
        int numberOfMsg;

        /*
         * Constructor takes an XML formatted string and parses the
         * string to create an OMSMessage object
         */
        public OMSMessage(String xmsData)
```

```
        {
            loadMessage(xmsData);

        }

        /*
         * Go through the XML document and extract the nodes to populate
the
         * datastructure.
         */
        private void loadMessage(String xmsData)
        {
            XmlDocument xmlLd;
            XmlNodeList nodeList;
            XmlNamespaceManager nsmgr;

            try
            {
                xmlLd = new XmlDocument();
                xmlLd.LoadXml(xmsData);

                nsmgr = new XmlNamespaceManager(xmlLd.NameTable);
                nsmgr.AddNamespace("xu",
"http://schemas.microsoft.com/office/Outlook/2006/OMS");
                nodeList = xmlLd.SelectNodes("/xu:xmsData", nsmgr);

                foreach (XmlNode node in nodeList)
                {
                    client = node.Attributes.GetNamedItem("client").Value;

                    for (int i = 0; i < node.ChildNodes.Count; i++)
                    {
                        switch (node.ChildNodes.Item(i).Name)
                        {
                            case "user":

loadUser(node.ChildNodes.Item(i).OuterXml);
                                break;
                            case "xmsHead":

loadHeader(node.ChildNodes.Item(i).OuterXml);
                                break;
                            case "xmsBody":

loadBody(node.ChildNodes.Item(i).OuterXml);
                                break;
                        }

                    }
                }

            }
            catch (Exception e)
            {
                throw;
```

```
            }
        }

        /*
         * Parse out the user portion of the XML document
         */
        private void loadUser(String xmsUser)
        {
            XmlDocument xmlLd;
            XmlNodeList nodeList;
            XmlNamespaceManager nsmgr;

            try
            {
                xmlLd = new XmlDocument();
                xmlLd.LoadXml(xmsUser);

                nsmgr = new XmlNamespaceManager(xmlLd.NameTable);
                nsmgr.AddNamespace("xu",
"http://schemas.microsoft.com/office/Outlook/2006/OMS");
                nodeList = xmlLd.SelectNodes("/xu:user",nsmgr);

                foreach (XmlNode node in nodeList)
                {
                    for (int i = 0; i < node.ChildNodes.Count; i++)
                    {
                        switch (node.ChildNodes.Item(i).Name)
                        {
                            case "userId":
                                userID =
node.ChildNodes.Item(i).InnerText;
                                break;
                            case "password":
                                password =
node.ChildNodes.Item(i).InnerText;
                                break;
                            case "replyPhone":
                                replyPhone =
node.ChildNodes.Item(i).InnerText;
                                break;
                            case "customData":
                                customData =
node.ChildNodes.Item(i).InnerText;
                                break;
                        }


                    }
                }

            }
            catch (Exception e)
            {
                throw;
            }
```

```
        }

        /*
         * Parsee out the header section of the XML document
         */
        private void loadHeader(String xmsHead)
        {
            XmlDocument xmlLd;
            XmlNodeList nodeList;
            XmlNamespaceManager nsmgr;

            try
            {
                xmlLd = new XmlDocument();
                xmlLd.LoadXml(xmsHead);

                nsmgr = new XmlNamespaceManager(xmlLd.NameTable);
                nsmgr.AddNamespace("xu",
"http://schemas.microsoft.com/office/Outlook/2006/OMS");
                nodeList = xmlLd.SelectNodes("/xu:xmsHead",nsmgr);

                foreach (XmlNode node in nodeList)
                {
                    for (int i = 0; i < node.ChildNodes.Count; i++)
                    {
                        switch (node.ChildNodes.Item(i).Name)
                        {
                            case "requiredService":
                                requiredService =
node.ChildNodes.Item(i).InnerText;
                                break;
                            case "sourceType":
                                sourceType =
node.ChildNodes.Item(i).InnerText;
                                break;
                            case "to":
                                //Initilize and populate array of
recipientList
                                recipientList = new
String[node.ChildNodes.Item(i).ChildNodes.Count];
                                for (int x = 0; x <
node.ChildNodes.Item(i).ChildNodes.Count; x++)
                                {
                                    recipientList[x] =
node.ChildNodes.Item(i).ChildNodes.Item(x).InnerText;
                                }
                                break;
                        }


                    }
                }

            }
            catch (Exception e)
```

```
            {
                throw;
            }
        }

        /*
         * Parse out the body section of the XML document.
         */
        private void loadBody(String xmsBody)
        {
            XmlDocument xmlLd;
            XmlNamespaceManager nsmgr;
            XmlNode node;
            try
            {
                xmlLd = new XmlDocument();
                xmlLd.LoadXml(xmsBody);

                nsmgr = new XmlNamespaceManager(xmlLd.NameTable);
                nsmgr.AddNamespace("xu",
"http://schemas.microsoft.com/office/Outlook/2006/OMS");

                node = xmlLd.SelectSingleNode("/xu:xmsBody", nsmgr);

                msgContent = node.Attributes.GetNamedItem("format").Value;
                message = new String[node.ChildNodes.Count,4];
                numberOfMsg = node.ChildNodes.Count;
                for (int i = 0; i < node.ChildNodes.Count; i++)
                {
                    if (node.ChildNodes.Item(i).Name=="content")
                    {
                        message[i,0] =
node.ChildNodes.Item(i).Attributes.GetNamedItem("contentType").Value;
                        message[i,1] =
node.ChildNodes.Item(i).Attributes.GetNamedItem("contentId").Value;
                        message[i,2] =
node.ChildNodes.Item(i).Attributes.GetNamedItem("contentLocation").Value;
                        message[i,3] = node.ChildNodes.Item(i).InnerText;
                    }
                }
            }
            catch (Exception e)
            {
                throw;
            }
        }

        /*
         * Store an instance of the sent SMS message in the log table
         */
        private void logMessage(int u_id)
        {
            SqlCommand cmd = null;

            try
```

```
            {
                cmd = new SqlCommand();
                cmd.Connection = Utility.getDBConnection();
                cmd.CommandType = System.Data.CommandType.StoredProcedure;
                cmd.CommandText = "logMessage";
                cmd.Parameters.Add("@u_id",
System.Data.SqlDbType.Int).Value = u_id;
                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                cmd.Connection.Close();
            }
            catch (Exception e)
            {
                throw;
            }
            finally
            {
                if (cmd != null)
                {
                    cmd.Dispose();
                }
            }
        }

        /*
         * Prepair and send message
         */
        public String sendMessage()
        {
            StringWriter sw = null;
            XmlTextWriter xw = null;
            OMSUser ou;
            int remainingMsg, numMsg;
            String ret;
            String countryCode = "001";
            try
            {
                sw = new StringWriter(new StringBuilder());
                xw = new XmlTextWriter(sw);

                ou = new OMSUser(userID, password);

                xw.WriteStartElement("xmsResponse",
"http://schemas.microsoft.com/office/Outlook/2006/OMS");
                if (!ou.validateUser())
                {
                    Utility.BuildError(xw, "invalidUser", true, "", "");
                    return sw.GetStringBuilder().ToString();
                }

                remainingMsg = ou.maxMessagesPerDay() -
ou.getDaysMessages();
                numMsg = recipientList.Length * numberOfMsg;

                if ((remainingMsg - numMsg) < 0)
```

```
                {
                    Utility.BuildError(xw, "perDayMsgLimit", true, "",
"");
                    return sw.GetStringBuilder().ToString();
                }

                if (MobedInterface.sendSMS(recipientList, numberOfMsg,
message, ou.getU_ID()))
                    Utility.BuildError(xw, "deliveryError", true, "", "");
                else
                {
                    Utility.BuildError(xw, "ok", false, "", "");
                    logMessage(ou.getU_ID());
                }

                xw.WriteEndElement();

                ret = sw.GetStringBuilder().ToString();
            }
            catch (Exception)
            {
                throw;
            }
            finally
            {
                xw.Close();
                sw.Close();
            }

            return ret;

        }
    }
}
```

OMSUser.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Xml;
using System.Data.SqlClient;
using System.Data;


namespace oms_lib
{

    /*
     * This class defines the datastructure for users and provides
     * the processes needed to parse the user.
```

```
     */
    public class OMSUser
    {
        String client;
        public String userID, password, customData;
        public int u_id, maxmessages, sp_id;
        public String fname, mname, lname, areacode, mobilenumber, email,
code;
        Boolean accepted_terms, replytomobile, replytomail, validUser;

        /*
         * Constructor takes an XML formatted string and parses the
         * string to create an OMSUser object
         */
        public OMSUser(String xmsUser)
        {
            loadUser(xmsUser);
            validUser = validateUser();
            if (validUser)
            {

                getUserInfo(userID);
            }
        }

        /*
         * Constructor takes a userID string and password string then
extracts
         * the user information from the database
         */
        public OMSUser(String _userID, String _password)
        {
            password = _password;
            userID = _userID;

            getUserInfo(userID);
        }


        /*
         * Validate the user information passed in through the constructor
         */
        public Boolean validateUser()
        {
            Boolean ret;
            ret = Utility.isValidUser(userID, password);
            return ret;
        }

        /*
         * Parse the user infomation from the XML string
         */
        private void loadUser(String xmsUser)
        {
            XmlDocument xmlLd;
```

```
            XmlNodeList nodeList;
            XmlNamespaceManager nsmgr;

            try
            {
                xmlLd = new XmlDocument();
                xmlLd.LoadXml(xmsUser);

                nsmgr = new XmlNamespaceManager(xmlLd.NameTable);
                nsmgr.AddNamespace("xu",
"http://schemas.microsoft.com/office/Outlook/2006/OMS");
                nodeList = xmlLd.SelectNodes("/xu:xmsUser", nsmgr);

                foreach(XmlNode node in nodeList)
                {
                    client = node.Attributes.GetNamedItem("client").Value;

                    for (int i = 0; i < node.ChildNodes.Count; i++)
                    {
                        switch (node.ChildNodes.Item(i).Name)
                        {
                            case "userId":
                                userID =
node.ChildNodes.Item(i).InnerText;
                                break;
                            case "password":
                                password =
node.ChildNodes.Item(i).InnerText;
                                break;
                            case "customData":
                                customData =
node.ChildNodes.Item(i).InnerText;
                                break;
                        }


                    }
                }

            }
            catch (Exception e)
            {
                throw;
            }
        }

        /*
         * Connect to the database and extract the user information
         */
        private void getUserInfo(String userID){
            SqlConnection conn = new SqlConnection();
            SqlDataAdapter sda = new SqlDataAdapter();
            DataSet ds = new DataSet();

            try
```

```
            {
                conn = Utility.getDBConnection();
                sda = new SqlDataAdapter("getUser", conn);
                sda.SelectCommand.CommandType =
System.Data.CommandType.StoredProcedure;
                sda.SelectCommand.Parameters.Add("@username",
SqlDbType.VarChar).Value = userID;

                sda.Fill(ds, "User");

                u_id = (int)ds.Tables["User"].Rows[0]["u_id"];
                fname =
(String)ds.Tables["User"].Rows[0]["fname"].ToString();
                mname =
(String)ds.Tables["User"].Rows[0]["mname"].ToString();
                lname =
(String)ds.Tables["User"].Rows[0]["lname"].ToString();
                areacode =
(String)ds.Tables["User"].Rows[0]["areacode"].ToString();
                mobilenumber =
(String)ds.Tables["User"].Rows[0]["mobilenumber"].ToString();
                email =
(String)ds.Tables["User"].Rows[0]["email"].ToString();
                accepted_terms =
(Boolean)ds.Tables["User"].Rows[0]["accepted_terms"];
                replytomail =
(Boolean)ds.Tables["User"].Rows[0]["replytomobile"];
                replytomobile =
(Boolean)ds.Tables["User"].Rows[0]["replytomail"];
                maxmessages =
(int)ds.Tables["User"].Rows[0]["maxmessages"];
                code =
(String)ds.Tables["User"].Rows[0]["code"].ToString();
                sp_id = (int)ds.Tables["User"].Rows[0]["sp_id"];

            }
            catch (Exception e)
            {
                throw;
            }
            finally
            {
                sda.Dispose();
                conn.Dispose();
            }
        }

        public static String save(Int32 ou_id, String fname, String mname,
String lname, String usrname, String password, String areacode, String
mobilenumber, String email, Int32 msgperday, Int32 sp_id)
        {
            SqlConnection conn = new SqlConnection();
            SqlCommand cmd = null;
            String errMsg = null;
            try
```

```
            {
                conn = oms_lib.Utility.getDBConnection();
                cmd = new SqlCommand("updateUser", conn);
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.Add("@ou_id", SqlDbType.Int).Value = ou_id;
                cmd.Parameters.Add("@fname", SqlDbType.VarChar).Value =
fname.ToString();
                cmd.Parameters.Add("@mname", SqlDbType.VarChar).Value =
mname.ToString();
                cmd.Parameters.Add("@lname", SqlDbType.VarChar).Value =
lname.ToString();
                cmd.Parameters.Add("@usrname", SqlDbType.VarChar).Value =
usrname.ToString();
                cmd.Parameters.Add("@password", SqlDbType.VarChar).Value =
password.ToString();
                cmd.Parameters.Add("@areacode", SqlDbType.VarChar).Value =
areacode.ToString();
                cmd.Parameters.Add("@mobilenumber",
SqlDbType.VarChar).Value = mobilenumber.Replace("-", "");
                cmd.Parameters.Add("@email", SqlDbType.VarChar).Value =
email.ToString();
                cmd.Parameters.Add("@msgperday", SqlDbType.Int).Value =
msgperday;
                cmd.Parameters.Add("@sp_id", SqlDbType.Int).Value = sp_id;

                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                errMsg = null;
            }
            catch (Exception)
            {
                errMsg = "Saved Failed";
            }
            finally
            {
                if (conn != null)
                {
                    if (conn.State != ConnectionState.Closed)
                        conn.Close();

                    conn.Dispose();
                }
            }

            return errMsg;
        }

        /*
         * Build the XML userInfo string that's returned to the
         * Outlook Mobile Services client.
         */
        public String getXMLString()
        {
            StringWriter sw = null;
```

```
            XmlWriter xw = null;

            try
            {
                sw = new StringWriter(new StringBuilder());
                xw = new XmlTextWriter(sw);
                xw.WriteStartElement("userInfo",
"http://schemas.microsoft.com/office/Outlook/2006/OMS");
                if (validUser)
                {
                    xw.WriteStartElement("replyPhone");
                    xw.WriteString(areacode + mobilenumber);
                    xw.WriteEndElement();

                    xw.WriteStartElement("smtpAddress");
                    xw.WriteString(email);
                    xw.WriteEndElement();

                    xw.WriteStartElement("error");
                    xw.WriteAttributeString("code", "ok");
                    xw.WriteEndElement();
                }
                else
                {
                    xw.WriteStartElement("error");
                    xw.WriteAttributeString("code", "invalidUser");
                    xw.WriteAttributeString("severity", "failure");
                    xw.WriteEndElement();
                }
                xw.WriteEndElement();

                return sw.GetStringBuilder().ToString();
            }
            catch (Exception e)
            {
                throw;
            }
        }

        /*
         * Get the number of messages sent for the day by the current user
         */
        public int getDaysMessages()
        {

            SqlCommand cmd =null;
            int ret;

            try
            {
                cmd = new SqlCommand("getDaysMessage",
Utility.getDBConnection());

                cmd.CommandType = System.Data.CommandType.StoredProcedure;
```

```
                cmd.Parameters.Add("@u_id", SqlDbType.VarChar).Value =
u_id;
                cmd.Parameters.Add("ret", SqlDbType.Int).Direction =
ParameterDirection.ReturnValue;

                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                ret = (int)cmd.Parameters["ret"].Value;
                cmd.Connection.Close();

            }
            catch (Exception e)
            {
                throw;
            }
            finally
            {
                cmd.Dispose();

            }
            return ret;
        }

        public int maxMessagesPerDay()
        {
            return maxmessages;
        }

        public int getU_ID()
        {

            return u_id;
        }

    }
}
```

Utility.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.IO;
using System.Xml.Schema;
using System.Data.SqlClient;
using System.Configuration;
using System.Net.Mail;
using System.Net;

namespace oms_lib
{
```

```
    public class Utility
    {
        /*
         * LoadXmlFromFile loads the file located in the location
         * specified by strFile
         */
        public static String LoadXmlFromFile(string strFile)
        {
            XmlDocument xdoc = new XmlDocument();
            String path = System.AppDomain.CurrentDomain.BaseDirectory;
            try
            {
                xdoc.Load(path + strFile);
            }
            catch (Exception e)
            {
                Utility.logError("getCarrierId:" + e.Message);
            }
            return xdoc.OuterXml;
        }

        /*
         * ValidateXMLString validates the XML in xmlDoc against
         * a schema definition file located in the xsdPath.
         */
        public static Boolean ValidateXMLString(String xmlDoc, String
xsdPath)
        {
            Boolean ret = false;
            String path = System.AppDomain.CurrentDomain.BaseDirectory +
"xsd\\";
            XmlReaderSettings settings = new XmlReaderSettings();

settings.Schemas.Add("http://schemas.microsoft.com/office/Outlook/2006/OMS
", path+xsdPath);
            settings.ValidationType = ValidationType.Schema;

            StringReader sr = new StringReader(xmlDoc);
            XmlReader reader = XmlReader.Create(sr,settings);
            XmlDocument document = new XmlDocument();
            try
            {
                document.Load(reader);
                ret = true;
            }
            catch (XmlSchemaValidationException xmle)
            {
                ret = false;
            }
            catch (Exception e)
            {
                throw;
            }
            return ret;
```

```
        }

        /*
         * Prep string before sending to database
         */
        public static String cleanString(Object str)
        {
            String ret = str.ToString();
            if (str == null)
                ret = "";
            return ret;

        }

        /*
         * This was used during development to save a file in the location
specified.
         */
        public static void SaveLogFile(string strErr, string strFileName)
        {
            System.IO.TextWriter tw = new
System.IO.StreamWriter("c:\\Temp\\" + strFileName);
            tw.WriteLine(strErr);

            tw.Close();

        }

        /*
         * Create a database connection and return it to the calling
procedure
         */
        public static SqlConnection getDBConnection()
        {
            SqlConnection conn;

            SqlConnectionStringBuilder csb = new
SqlConnectionStringBuilder();

            csb.Password = "oms";
            csb.UserID = "oms";
            csb.InitialCatalog = "oms";
            csb.DataSource = "msdev";
            conn = new SqlConnection(csb.ConnectionString);
            try
            {
                conn.Open();
            }
            catch(Exception e)
            {
                throw;
            }
            finally
            {
                if(conn.State == System.Data.ConnectionState.Open){
```

```
                    conn.Close();
                }
            }

            return conn;
        }

        /*
         * Verify that the username and password exist in the system and
are valid.
         */
        public static Boolean isValidUser(String userName, String
password)
        {
            SqlCommand cmd = null;
            Boolean ret = false;
            int flag = 0;
            try
            {
                cmd = new SqlCommand();
                cmd.Connection = getDBConnection();
                cmd.CommandType = System.Data.CommandType.StoredProcedure;
                cmd.CommandText = "isValidUser";
                cmd.Parameters.Add("@userName",
System.Data.SqlDbType.VarChar).Value=userName;
                cmd.Parameters.Add("@password",
System.Data.SqlDbType.VarChar).Value=password;
                cmd.Parameters.Add("ret",
System.Data.SqlDbType.Bit).Direction =
System.Data.ParameterDirection.ReturnValue;
                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                cmd.Connection.Close();

                ret = (Boolean)cmd.Parameters["ret"].Value;
            }
            catch (Exception e)
            {
                throw;
            }
            finally
            {
                if (cmd != null)
                    cmd.Dispose();
            }
            return ret;
        }

        /*
         * Builds an xml formatted string that is returned to the Outlook
client when errors occur during processing
         */
        public static void BuildError(XmlTextWriter wr,   string errCode,
bool bFailed,   string strContent,   string strRecipients)
        {
```

```
            wr.WriteStartElement("error");
            wr.WriteAttributeString("code", errCode);
            if(bFailed)
                wr.WriteAttributeString("severity", "failure");

            if (strContent.Length > 0)
            {
                wr.WriteStartElement("content");
                wr.WriteString(strContent);
                wr.WriteEndElement(); // </content>
            }
            if (strRecipients.Length > 0)
            {
                wr.WriteStartElement("recipientList");
                wr.WriteString(strRecipients);
                wr.WriteEndElement(); // </recipientList>
            }
            wr.WriteEndElement(); // </error>
        }

        /*
         * This adds an entry in the Applicaction log table to help in
         * debugging
         */
        public static void logError(int u_id, String message)
        {
            SqlCommand cmd = null;

            try
            {
                cmd = new SqlCommand();
                cmd.Connection = Utility.getDBConnection();
                cmd.CommandType = System.Data.CommandType.StoredProcedure;
                cmd.CommandText = "logError";
                cmd.Parameters.Add("@u_id",
System.Data.SqlDbType.Int).Value = u_id;
                cmd.Parameters.Add("@message",
System.Data.SqlDbType.VarChar).Value = message;

                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                cmd.Connection.Close();
            }
            catch (Exception e)
            {
                throw;
            }
            finally
            {
                cmd.Dispose();
            }
        }

        /*
         * This adds an entry in the Applicaction log table to help in
```

```
         * debugging
         */
        public static void logError(String message)
        {
            SqlCommand cmd = null;

            try
            {
                cmd = new SqlCommand();
                cmd.Connection = Utility.getDBConnection();
                cmd.CommandType = System.Data.CommandType.StoredProcedure;
                cmd.CommandText = "logError";
                cmd.Parameters.Add("@u_id",
System.Data.SqlDbType.Int).Value = System.DBNull.Value;
                cmd.Parameters.Add("@message",
System.Data.SqlDbType.VarChar).Value = message;

                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                cmd.Connection.Close();
            }
            catch (Exception e)
            {
                throw;
            }
            finally
            {
                cmd.Dispose();
            }
        }

        /*
         * Send an email message
         */
        public static void sendMail(String from, String email, String
message)
        {
            MailMessage mail = null;
            try
            {
                MailAddress frm = new MailAddress(from+"@uncw-oms.com",
from);
                MailAddress to = new MailAddress(email);

                mail = new MailMessage(frm, to);

                /* These headers are specific to OMS and enables to the
Outlook
                 * client to process these messages as text messages
                 */
                mail.Headers.Add("Content-class", "MS-OMS-SMS");
                mail.Headers.Add("X-MS-Reply-to-mobile", from);


                mail.Body = message;
```

```
                SmtpClient s = new SmtpClient("smtp-server.ec.rr.com");
                s.Credentials = CredentialCache.DefaultNetworkCredentials;
                s.Send(mail);

            }
            catch (Exception)
            {
                throw;
            }
        }

        /*
         * Extracts the dest. number and gets the matching email
         * adddress
         */
        public static String getEmailAddress(String message)
        {
            String email = "", tonum = "", msg = message;

            try
            {

                tonum = msg.Substring(msg.ToUpper().IndexOf("OMS ")+4);
                tonum = tonum.Substring(0,10);

                SqlCommand cmd = null;


                try
                {
                    cmd = new SqlCommand("getEmailAddress",
Utility.getDBConnection());

                    cmd.CommandType =
System.Data.CommandType.StoredProcedure;
                    cmd.Parameters.Add("@phoneNumber",
System.Data.SqlDbType.VarChar).Value = tonum;
                    cmd.Parameters.Add("ret",
System.Data.SqlDbType.VarChar).Direction =
System.Data.ParameterDirection.ReturnValue;

                    cmd.Connection.Open();
                    cmd.ExecuteNonQuery();
                    email = cmd.Parameters["ret"].Value.ToString();
                    cmd.Connection.Close();

                }
                catch (Exception e)
                {
                    return "";
                }
                finally
                {
                    cmd.Dispose();
```

```
                }

            }
            catch (Exception)
            {
                throw;
            }
            return email;
        }

        public static Boolean createUser(String userName, String password,
String email, String firstName, String lastName, String areaCode, String
mobileNumber, String carrierID, String errMsg)
        {
            SqlConnection conn = new SqlConnection();
            SqlCommand cmd;

            try
            {
                conn = Utility.getDBConnection();
                cmd = new SqlCommand("createUser", conn);
                cmd.CommandType = System.Data.CommandType.StoredProcedure;

                cmd.Parameters.Add("@username",
System.Data.SqlDbType.VarChar).Value = userName;
                cmd.Parameters.Add("@password",
System.Data.SqlDbType.VarChar).Value = password;
                cmd.Parameters.Add("@email",
System.Data.SqlDbType.VarChar).Value = email;
                cmd.Parameters.Add("@firstname",
System.Data.SqlDbType.VarChar).Value = firstName;
                cmd.Parameters.Add("@lastname",
System.Data.SqlDbType.VarChar).Value = lastName;
                cmd.Parameters.Add("@areacode",
System.Data.SqlDbType.VarChar).Value = areaCode;
                cmd.Parameters.Add("@mobilenumber",
System.Data.SqlDbType.VarChar).Value = mobileNumber.Replace("-","");
                cmd.Parameters.Add("@carrierid",
System.Data.SqlDbType.VarChar).Value = carrierID;
                cmd.Parameters.Add("@terms",
System.Data.SqlDbType.VarChar).Value = '1';


                cmd.Parameters.Add("ret",
System.Data.SqlDbType.VarChar).Direction =
System.Data.ParameterDirection.ReturnValue;
                cmd.Connection.Open();
                cmd.ExecuteNonQuery();
                cmd.Connection.Close();
                //errMsg = (String)cmd.Parameters["ret"].Value;
            }
            catch (Exception e)
            {
                errMsg = "Saved Failed";
```

```
                return false;
            }
            finally
            {
                conn.Dispose();
            }

            errMsg = "";
            return true;
        }

        public static Boolean checkUniqueValues(String table, String
column, String value)
        {
            SqlConnection conn = new SqlConnection();
            SqlCommand cmd  = null;
            SqlDataAdapter sda = null;
            Boolean ret;
            String sql;
            int cnt;
            System.Data.DataSet ds = new System.Data.DataSet();
            try
            {
                sql = "select count("+column.ToString()+") cnt from
"+table.ToString()+" where "+column.ToString()+" =
'"+value.ToString()+"'";
                conn = Utility.getDBConnection();
                cmd = new SqlCommand(sql, conn);
                sda = new SqlDataAdapter(cmd);

                sda.Fill(ds);

                cnt = (int)ds.Tables[0].Rows[0][0];
                if (cnt == 0)
                    ret = true;
                else
                    ret = false;
            }
            catch (Exception e)
            {
                ret = false;
            }
            finally
            {
                if (cmd != null)
                {
                    cmd.Dispose();
                }
            }
            return ret;
        }
    }

    public class oms_exception : Exception
    {
```

```
        public oms_exception(string message) : base(message)
        {

        }

    }
}
```

Oms-service Project

Oms-uncw.asmx.cs

```
using System;
using System.Data;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.ComponentModel;
using oms_lib;

namespace oms_service
{
    /*
     * The oms_uncw class is a webservice with one WebMethod
     * This class is responsible for processing SOAP and sending
     * the messages to an Outlook client.
     */
    [WebService(Namespace = "http://oms-uncw.homeip.net/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    public class oms_uncw : System.Web.Services.WebService
    {

        /*
         * SMStoEmail - web method
         * Parameters
         *  from - Is the mobile number of the sending device
         *  message - This is the conent of the SMS message.
         *            The message must begin with "OMS <destination mobile
number> <message text>
         *            ex OMS 0011115555555 This is a test message.
         * Return - "Success" is returned if the method executes correctly
and the exception
         *          message is returned otherwise.
         */
        [System.Web.Services.WebMethod()]
        public string SMStoEmail(String from, String message)
```

```
        {

            String frm, email, ret = "Success";
            try
            {
                //If the from number has 13 characters then remove the
first 3 chars
                if (from.Length == 13)
                    frm = from.Substring(3);
                else
                    frm = from;


                email = Utility.getEmailAddress(message);


                Utility.sendMail(frm, email, message);
            }
            catch (Exception e)
            {
                Utility.logError("SMStoEmail:" + e.Message + " from: " +
from + " message: " + message);
                Utility.logError("SMStoEmail:" + e.StackTrace);
                ret = "Error: " + e.Message;
            }
            return ret;
        }



    }
}
```

Oms.asmx.cs

```
using System;
using System.Data;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.ComponentModel;
using oms_lib;
using System.Xml;


namespace oms_service
{

    /*
     * This class is utilized by the Outlook client to send sms messages
     * to mobile devices.
```

```
     */
    [WebService(Namespace =
"http://schemas.microsoft.com/office/Outlook/2006/OMS")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    public class oms : System.Web.Services.WebService
    {

        /*
         * Gets OMS Server settings
         */
        [System.Web.Services.WebMethod()]
        public string GetServiceInfo()
        {
            Boolean valid=false;
            String serviceInfo="";
            try
            {
                serviceInfo = Utility.LoadXmlFromFile("serviceInfo.xml");
                valid = Utility.ValidateXMLString(serviceInfo,
"serviceInfo.xsd");
            }
            catch (Exception e)
            {
                Utility.logError("GetServiceInfo:"+e.Message);
            }
            if (valid)
            {
                return serviceInfo;
            }
            else //invalid XML Schema
                return "";
        }

        /*
         * Takes an XML formatted useer object which is generated and
passed in from the
         * outlook client and validates the user then extracts the related
user information
         */
        [System.Web.Services.WebMethod()]
        public string GetUserInfo(String xmsUser)
        {
            String xmsUserRet="";
            try
            {

                if (!Utility.ValidateXMLString(xmsUser, "xmsUser.xsd"))
                    return "";

                OMSUser usr;
                usr = new OMSUser(xmsUser);

                xmsUserRet = usr.getXMLString();
```

```
                if (!Utility.ValidateXMLString(xmsUserRet,
"userInfo.xsd"))
                    return "";
            }
            catch (Exception e)
            {
                Utility.logError("GetUserInfo:" + e.Message);
                throw;
            }
            //Utility.SaveLogFile(xmsUserRet, "xmsUserRet.txt");
            return xmsUserRet;
        }

        /*
         * Takes an XML formatted string which contains all the related
user information and
         * SMS message.  This method then parses the message and sends it
to MOBED for delivery.
         */
        [System.Web.Services.WebMethod()]
        public string SendXms(String xmsData)
        {
            String ret;
            OMSMessage msg = null;
            try
            {

                //Utility.SaveLogFile(xmsData, "XmsData.xml");
                if (!Utility.ValidateXMLString(xmsData, "xmsData.xsd"))
                    return "<?xml version=\"1.0\" encoding=\"utf-16\"?> "
+
                           "<xmsResponse
xmlns=\"http://schemas.microsoft.com/office/Outlook/2006/OMS\">"+
                              "<error code=\"invalid message format\"
severity=\"failure\">"+
                              "</error> "+
                           "</xmsResponse>";

                msg = new OMSMessage(xmsData);

                ret = msg.sendMessage();
            }
            catch (Exception e)
            {
                Utility.logError("SendXms:" + e.Message);
                throw;
            }

            return ret;
        }


    }
}
```

Web.config

```
<?xml version="1.0"?>

<configuration>

    <configSections>
    </configSections>
  <appSettings>
    <add key="dbsource" value="msdev"/>
    <add key="username" value="oms"/>
    <add key="pass" value="oms"/>
  </appSettings>
    <connectionStrings/>

    <system.web>

        <compilation debug="true" />

        <authentication mode="Windows" />

    </system.web>
</configuration>
```

Signup Project

Create.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

namespace signup
{
    public partial class Create1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (this.IsPostBack == true)
```

```
                return;
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            String errMsg;

            errMsg = "";
            if (checkUniqueValues(UserName.Text.ToString(),
AreaCode.Text.ToString(), PhoneNumber.Text.ToString()))
                if (!oms_lib.Utility.createUser(UserName.Text.ToString(),
Password.Text.ToString(), Email.Text.ToString(),
FirstName.Text.ToString(), LastName.Text.ToString(),
AreaCode.Text.ToString(), PhoneNumber.Text.ToString(),
Carrier.SelectedValue, errMsg))
                {
                    ErrorMessage.Text = "Saved Failed";

                }
                else
                {
                    ErrorMessage.Text = "";
                    Response.Redirect("default.aspx");
                }


        }

        protected Boolean checkUniqueValues(String userName, String
areaCode, String mobileNumber)
        {
            String tableName = "oms_user";
            Boolean ret = true;

            try
            {
                if (!oms_lib.Utility.checkUniqueValues(tableName,
"username", userName.ToString()))
                {
                    ret = false;
                    ErrorMessage.Text = "Username already used " ;
                }
            }
            catch (Exception e)
            {
                ErrorMessage.Text += "Username already used ";
                ret = false;
            }

            try
            {
                if (!oms_lib.Utility.checkUniqueValues(tableName,
"areacode+mobilenumber", areaCode.ToString()+mobileNumber.ToString()))
                {
                    ret = false;
```

```
                    ErrorMessage.Text += "Area code/Phone number already
used ";
                }
            }
            catch (Exception e)
            {
                ErrorMessage.Text += "Area code/Phone number already used
";
                ret = false;
            }
            return ret;
        }
    }
}
```

Create.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Create.aspx.cs"
Inherits="signup.Create1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <img src="OMS2.gif" />
        <table border="0" style="font-size: 100%; font-family:
Verdana;background-color:#F7F6F3;border:1px solid #E6E2D8;Font-
Size:0.8em">
                            <tr>
                                <td align="center" colspan="2"
style="font-weight: bold; color: white; background-color: #5d7b9d">
                                    Sign Up for Your New Account</td>
                            </tr>
                            <tr>
                                <td align="right">
                                    <asp:Label ID="UserNameLabel"
runat="server" AssociatedControlID="UserName">User Name:</asp:Label></td>
                                <td>
                                    <asp:TextBox ID="UserName"
runat="server"></asp:TextBox>
                                    <asp:RequiredFieldValidator
ID="UserNameRequired" runat="server" ControlToValidate="UserName"
```

```
                                        ErrorMessage="User Name is
required." ToolTip="User Name is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                    </td>
                                </tr>
                                <tr>
                                    <td align="right">
                                        <asp:Label ID="PasswordLabel"
runat="server" AssociatedControlID="Password">Password:</asp:Label></td>
                                    <td>
                                        <asp:TextBox ID="Password"
runat="server" TextMode="Password"></asp:TextBox>
                                        <asp:RequiredFieldValidator
ID="PasswordRequired" runat="server" ControlToValidate="Password"
                                            ErrorMessage="Password is
required." ToolTip="Password is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                    </td>
                                </tr>
                                <tr>
                                    <td align="right">
                                        <asp:Label ID="ConfirmPasswordLabel"
runat="server" AssociatedControlID="ConfirmPassword">Confirm
Password:</asp:Label></td>
                                    <td>
                                        <asp:TextBox ID="ConfirmPassword"
runat="server" TextMode="Password"></asp:TextBox>
                                        <asp:RequiredFieldValidator
ID="ConfirmPasswordRequired" runat="server"
ControlToValidate="ConfirmPassword"
                                            ErrorMessage="Confirm Password is
required." ToolTip="Confirm Password is required."

ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                    </td>
                                </tr>
                                <tr>
                                    <td align="right">
                                        <asp:Label ID="EmailLabel"
runat="server" AssociatedControlID="Email">E-mail:</asp:Label></td>
                                    <td>
                                        <asp:TextBox ID="Email"
runat="server"></asp:TextBox>
                                        <asp:RequiredFieldValidator
ID="EmailRequired" runat="server" ControlToValidate="Email"
                                            ErrorMessage="E-mail is required."
ToolTip="E-mail is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                        <asp:RegularExpressionValidator
ID="EmailExp" runat="server" ControlToValidate="Email"
                                            ErrorMessage="Invalid e-mail
format." ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*"
```

```
ValidationGroup="CreateUserWizard1">*</asp:RegularExpressionValidator></td
>
                            </tr>
                            <tr>
                                <td align="right">
                                    <asp:Label ID="FirstNameLabel"
runat="server" AssociatedControlID="FirstName">First
Name:</asp:Label></td>
                                <td>
                                    <asp:TextBox ID="FirstName"
runat="server"></asp:TextBox>
                                    <asp:RequiredFieldValidator
ID="FirstNameRequired" runat="server" ControlToValidate="FirstName"
                                        ErrorMessage="First name is
required." ToolTip="First name is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                </td>
                            </tr>
                            <tr>
                                <td align="right">
                                    <asp:Label ID="AnswerLabel"
runat="server" AssociatedControlID="LastName">Last Name:</asp:Label></td>
                                <td>
                                    <asp:TextBox ID="LastName"
runat="server"></asp:TextBox>
                                    <asp:RequiredFieldValidator
ID="AnswerRequired" runat="server" ControlToValidate="LastName"
                                        ErrorMessage="Last name is
required." ToolTip="Last name is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                </td>
                            </tr>
                            <tr>
                                <td align="right">
                                    <asp:Label ID="Label1" runat="server"
AssociatedControlID="AreaCode">Mobile Number:</asp:Label></td>
                                <td>
                                    <asp:TextBox ID="AreaCode"
runat="server" Columns="3" MaxLength="3"></asp:TextBox>
                                    <asp:RequiredFieldValidator
ID="RequiredFieldValidator1" runat="server" ControlToValidate="AreaCode"
                                        ErrorMessage="Area code is
required." ToolTip="Area code is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                    <asp:RegularExpressionValidator
ID="RegularExpressionValidator2" runat="server"
ControlToValidate="AreaCode"
                                        Display="Dynamic"
ErrorMessage="Invalid area code format" ToolTip="invalid area code format"
                                        ValidationExpression="\d{3}"
ValidationGroup="CreateUserWizard1">*</asp:RegularExpressionValidator>
                                    <asp:TextBox ID="PhoneNumber"
runat="server" Columns="8" MaxLength="8"></asp:TextBox> 
```

```
                                    <asp:RequiredFieldValidator
ID="RequiredFieldValidator3" runat="server"
ControlToValidate="PhoneNumber"
                                        ErrorMessage="Phone number is
required." ToolTip="Phone number is required."

ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                    <asp:RegularExpressionValidator
ID="RegularExpressionValidator1" runat="server"
ControlToValidate="PhoneNumber"
                                        ErrorMessage="Invalid mobile
number format" ToolTip="Invalid mobile number format"
                                        ValidationExpression="\d{3}-
?\d{4}"
ValidationGroup="CreateUserWizard1">*</asp:RegularExpressionValidator>
                                </td>
                            </tr>
                            <tr>
                                <td align="right">
                                    <asp:Label ID="Label2" runat="server"
AssociatedControlID="Carrier">Carrier:</asp:Label></td>
                                <td>
                                    <asp:DropDownList ID="Carrier"
runat="server" DataSourceID="oms" DataTextField="name"
                                        DataValueField="sp_id">
                                    </asp:DropDownList>
                                    <asp:RequiredFieldValidator
ID="RequiredFieldValidator2" runat="server" ControlToValidate="Carrier"
                                        ErrorMessage="Carrier is
required." ToolTip="Carrier is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                    <asp:SqlDataSource ID="oms"
runat="server" ConnectionString="<%$ ConnectionStrings:OMSConnectionString
%>"
                                        SelectCommand="SELECT [sp_id],
[name] FROM [service_provider] ORDER BY [name]"></asp:SqlDataSource>
                                </td>
                            </tr>
                            <tr>
                                <td align="right">
                                </td>
                                <td>
                                     </td>
                            </tr>
                            <tr>
                                <td align="center" colspan="2"
style="height: 18px">
                                    <asp:CompareValidator
ID="PasswordCompare" runat="server" ControlToCompare="Password"

ControlToValidate="ConfirmPassword" Display="Dynamic" ErrorMessage="The
Password and Confirmation Password must match."

ValidationGroup="CreateUserWizard1"></asp:CompareValidator>
                                </td>
```

```
                            </tr>
                            <tr>
                                <td align="center" colspan="2"
style="color: red">
                                    <asp:Literal ID="ErrorMessage"
runat="server" EnableViewState="False"></asp:Literal>
                                </td>
                            </tr>
            <tr>
                <td align="center" colspan="2" style="color: red">
                    <asp:Label ID="ERR" runat="server" style="Font-
Size:0.8em;color: Red;"></asp:Label></td>
            </tr>
                            <tr align="right">
                                <td align="right" colspan="2"
style="padding:5px 5px 5px 5px;">
                                    <asp:Button ID="Button1"
runat="server" BackColor="#FFFBFF" BorderColor="#CCCCCC"
                                        BorderStyle="Solid"
BorderWidth="1px" Font-Names="Verdana"
                                        ForeColor="#284775" Text="Create
User" OnClick="Button1_Click" ValidationGroup="CreateUserWizard1" />
                                </td>
                            </tr>
                        </table>
    </form>
</body>
</html>
```

Default.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

namespace signup
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (this.IsPostBack)
                return;

            populateUserSettings();
```

```
        }

        protected void saveUser()
        {
            String errMsg = "";
            try
            {
                errMsg =
oms_lib.OMSUser.save(System.Convert.ToInt32(Session["ou_id"]),
txtFirstName.Text.ToString(), txtMiddleName.Text.ToString(),
txtLastName.Text.ToString(), txtUserName.Text.ToString(),
txtPassword.Text.ToString(), txtAreaCode.Text.ToString(),
txtMobileNumber.Text.ToString(), txtEmail.Text.ToString(),
System.Convert.ToInt32(txtMsgPerDay.Text.ToString()),
System.Convert.ToInt32(ddlServiceProvider.SelectedValue));

                oms_lib.OMSUser ou = new
oms_lib.OMSUser(txtUserName.Text.ToString(), txtPassword.Text.ToString());
                Session["ou"] = ou;
                populateUserSettings();
            }
            catch (Exception)
            {

            }
        }

        protected void populateUserSettings()
        {
            oms_lib.OMSUser ou;

            try
            {
                ou = (oms_lib.OMSUser)Session["ou"];

                txtFirstName.Text = ou.fname.ToString();
                txtMiddleName.Text = ou.mname.ToString();
                txtLastName.Text = ou.lname.ToString();

                txtUserName.Text = ou.userID.ToString();
                txtPassword.Text = ou.password.ToString();
                txtAreaCode.Text = ou.areacode.ToString();
                txtMobileNumber.Text = ou.mobilenumber.ToString().Trim();
                txtEmail.Text = ou.email.ToString();
                txtMsgPerDay.Text = ou.maxmessages.ToString();
                ddlServiceProvider.SelectedValue = ou.sp_id.ToString();

            }
            catch (Exception)
            {

            }
        }

        protected void Button1_Click(object sender, EventArgs e)
```

```
        {
            saveUser();
        }


    }
}
```

Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="signup._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
    <style type="text/css">
        .textBox
        {
            text-align:left;
            font-size:0.8em;
        }
        .label
        {

        }
    </style>
</head>
<body style="font-size:0.8em;">
    <form id="form1" runat="server">
    <div >
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:OMSConnectionString %>"
            SelectCommand="select pn.pn_id, sp.sp_id, pn.number, sp.name
from service_provider sp, phone_number pn where sp.sp_id = pn.sp_id and
pn.ou_id = @ou_id"
            UpdateCommand="UPDATE [phone_number] SET [sp_id] = @sp_id
where [pn_id] = @pn_id">
            <UpdateParameters>
                <asp:Parameter Name="sp_id" />
                <asp:Parameter Name="pn_id" />
            </UpdateParameters>
            <SelectParameters>
                <asp:SessionParameter DefaultValue="2" Name="ou_id"
SessionField="ou_id" />
            </SelectParameters>
        </asp:SqlDataSource>
```

```
        <asp:SqlDataSource ID="SqlDataSource2" runat="server"
ConnectionString="<%$ ConnectionStrings:OMSConnectionString %>"
            SelectCommand="SELECT [sp_id],[name] FROM [service_provider]
ORDER BY [name]"></asp:SqlDataSource>
        <img src="OMS2.gif" /> 
        <asp:LoginStatus ID="LoginStatus1" runat="server"
LogoutPageUrl="login.aspx" />
        <table style="width:1000px;">
            <tr>
                <td style="vertical-align:top;">
                    <asp:Panel ID="PhoneNumberList" runat="server"
Height="305px" Width="500px" ScrollBars="Auto" GroupingText="Phone Book"
BackColor="#F7F6F3">
                        <div style="height:280px;">
                        <asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="False" DataKeyNames="pn_id"
                            DataSourceID="SqlDataSource1"
AutoGenerateEditButton="True" CellPadding="4" ForeColor="#333333"
GridLines="None" Width="95%" AllowPaging="True">
                            <Columns>
                                <asp:BoundField DataField="number"
HeaderText="Number" SortExpression="number" ReadOnly="True" >
                                    <HeaderStyle HorizontalAlign="Left"
Width="100px" />
                                </asp:BoundField>
                                <asp:TemplateField HeaderText="Name"
SortExpression="name">
                                    <EditItemTemplate>
                                        <asp:DropDownList
ID="DropDownList1" runat="server" DataSourceID="SqlDataSource2"
                                            DataTextField="name"
DataValueField="sp_id" SelectedValue='<%# Bind("sp_id") %>'>
                                        </asp:DropDownList>
                                    </EditItemTemplate>
                                    <ItemTemplate>
                                        <asp:Label ID="Label1"
runat="server" Text='<%# Bind("name") %>' ></asp:Label>
                                    </ItemTemplate>
                                    <HeaderStyle HorizontalAlign="Left" />
                                </asp:TemplateField>

                            </Columns>
                            <FooterStyle BackColor="#5D7B9D" Font-
Bold="True" ForeColor="White" />
                            <RowStyle BackColor="#F7F6F3"
ForeColor="#333333" />
                            <EditRowStyle BackColor="#999999" />
                            <SelectedRowStyle BackColor="#E2DED6" Font-
Bold="True" ForeColor="#333333" />
                            <PagerStyle BackColor="#284775"
ForeColor="White" HorizontalAlign="Center" />
                            <HeaderStyle BackColor="#5D7B9D" Font-
Bold="True" ForeColor="White" />
                            <AlternatingRowStyle BackColor="White"
ForeColor="#284775" />
```

```
                        </asp:GridView>
                        </div>
                       </asp:Panel>
                       <br />
                    <asp:Panel ID="Panel1" runat="server" Height="125px"
Width="500px" BackColor="#F7F6F3" GroupingText="Messages Stats">
                        <asp:GridView ID="GridView2" runat="server"
AutoGenerateColumns="False" CellPadding="4"
                            DataSourceID="SqlDataSource4"
ForeColor="#333333" GridLines="None">
                            <FooterStyle BackColor="#5D7B9D" Font-
Bold="True" ForeColor="White" />
                            <Columns>
                                <asp:BoundField DataField="day"
HeaderText="Date" SortExpression="day" />
                                <asp:BoundField DataField="cnt"
HeaderText="# of Messages" SortExpression="cnt" />
                            </Columns>
                            <RowStyle BackColor="#F7F6F3"
ForeColor="#333333" />
                            <EditRowStyle BackColor="#999999" />
                            <SelectedRowStyle BackColor="#E2DED6" Font-
Bold="True" ForeColor="#333333" />
                            <PagerStyle BackColor="#284775"
ForeColor="White" HorizontalAlign="Center" />
                            <HeaderStyle BackColor="#5D7B9D" Font-
Bold="True" ForeColor="White" />
                            <AlternatingRowStyle BackColor="White"
ForeColor="#284775" />
                        </asp:GridView>
                        <asp:SqlDataSource ID="SqlDataSource4"
runat="server" ConnectionString="<%$ ConnectionStrings:OMSConnectionString
%>"
                            SelectCommand="SELECT
convert(varchar,message_date+3,101) day, count(ml_id) cnt &#13;&#10;FROM
[message_log] &#13;&#10;where u_id = @u_id&#13;&#10;and message_date
between convert(datetime,convert(varchar,getdate(),101))-3 and
getdate()&#13;&#10;group by
convert(varchar,message_date+3,101)&#13;&#10;order by
convert(varchar,message_date+3,101) desc">
                            <SelectParameters>
                                <asp:SessionParameter DefaultValue="2"
Name="u_id" SessionField="ou_id" />
                            </SelectParameters>
                        </asp:SqlDataSource>
                    </asp:Panel>
            </td>
            <td style="text-align:right;vertical-align:top;">
                <asp:Panel ID="UserSettings" runat="server" Height="435px"
Width="450px" ScrollBars="Auto" GroupingText="User Settings"
BackColor="#F7F6F3">
                    <div style="height:410px;">
                    <table style="width:100%">
                        <tr>
                            <td class="label">
```

```
                                <asp:Label ID="Label2" runat="server"
Text="First Name:"></asp:Label></td>
                            <td class="textBox"><asp:TextBox
ID="txtFirstName" runat="server" Columns="20"
MaxLength="20"></asp:TextBox>
                                <asp:RequiredFieldValidator
ID="FirstNameRequired" runat="server" ControlToValidate="txtFirstName"
                                    ErrorMessage="First name is required."
ToolTip="First name is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator></td>
                        </tr>
                        <tr>
                            <td class="label">
                                <asp:Label ID="Label3" runat="server"
Text="Middle Name:"></asp:Label></td>
                            <td class="textBox">
                                <asp:TextBox ID="txtMiddleName"
runat="server" Columns="20" MaxLength="20"></asp:TextBox></td>
                        </tr>
                        <tr>
                            <td class="label">
                                <asp:Label ID="Label4" runat="server"
Text="Last Name:"></asp:Label></td>
                            <td class="textBox">
                                <asp:TextBox ID="txtLastName"
runat="server" Columns="20" MaxLength="20"></asp:TextBox>
                                <asp:RequiredFieldValidator
ID="AnswerRequired" runat="server" ControlToValidate="txtLastName"
                                    ErrorMessage="Last name is required."
ToolTip="Last name is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator></td>
                        </tr>
                        <tr>
                            <td class="label">
                                <asp:Label ID="Label5" runat="server"
Text="User Name:"></asp:Label></td>
                            <td class="textBox">
                                <asp:TextBox ID="txtUserName"
runat="server" Columns="20" MaxLength="20"
ReadOnly="True"></asp:TextBox></td>
                        </tr>
                        <tr>
                            <td class="label">
                                <asp:Label ID="Label6" runat="server"
Text="Password:"></asp:Label></td>
                            <td class="textBox">
                                <asp:TextBox ID="txtPassword"
runat="server" Columns="20" MaxLength="20"
TextMode="Password"></asp:TextBox>
                                <asp:RequiredFieldValidator
ID="PasswordRequired" runat="server" ControlToValidate="txtPassword"
                                    ErrorMessage="Password is required."
ToolTip="Password is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator></td>
                        </tr>
```

```
                        <tr>
                            <td class="label">
                                <asp:Label ID="ConfirmPasswordLabel"
runat="server" AssociatedControlID="ConfirmPassword">Confirm
Password:</asp:Label></td>
                            <td class="textBox">
                                <asp:TextBox ID="ConfirmPassword"
runat="server" TextMode="Password"></asp:TextBox>
                                <asp:RequiredFieldValidator
                                    ID="ConfirmPasswordRequired"
runat="server" ControlToValidate="ConfirmPassword"
                                    ErrorMessage="Confirm Password is
required." ToolTip="Confirm Password is required."

ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator></td>
                        </tr>
                        <tr>
                            <td class="label">
                                <asp:Label ID="Label7" runat="server"
Text="Mobile Number:"></asp:Label></td>
                            <td class="textBox">
                                <asp:TextBox ID="txtAreaCode"
runat="server" Columns="20" MaxLength="3" Width="50px"></asp:TextBox>
                                <asp:RequiredFieldValidator
ID="RequiredFieldValidator1" runat="server"
ControlToValidate="txtAreaCode"
                                    ErrorMessage="Area code is required."
ToolTip="Area code is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                <asp:TextBox ID="txtMobileNumber"
runat="server" Columns="20" MaxLength="8" Width="80px"></asp:TextBox>
                                <asp:RequiredFieldValidator
ID="RequiredFieldValidator3" runat="server"
ControlToValidate="txtMobileNumber"
                                    ErrorMessage="Phone number is
required." ToolTip="Phone number is required."

ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator></td>
                        </tr>
                        <tr>
                            <td class="label">
                                <asp:Label ID="Label8" runat="server"
Text="Email:"></asp:Label></td>
                            <td class="textBox">
                                <asp:TextBox ID="txtEmail" runat="server"
Columns="20" MaxLength="20"></asp:TextBox>
                                <asp:RequiredFieldValidator
ID="EmailRequired" runat="server" ControlToValidate="txtEmail"
                                    ErrorMessage="E-mail is required."
ToolTip="E-mail is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator></td>
                        </tr>
                        <tr>
                            <td class="label">
```

```
                                <asp:Label ID="Label9" runat="server"
Text="Messages Per Day:"></asp:Label></td>
                            <td class="textBox">
                                <asp:TextBox ID="txtMsgPerDay"
runat="server" Columns="20" MaxLength="20"></asp:TextBox></td>
                        </tr>
                        <tr>
                            <td class="label">
                                <asp:Label ID="Label10" runat="server"
Text="Service Provider:"></asp:Label></td>
                            <td class="textBox">
                                <asp:DropDownList ID="ddlServiceProvider"
runat="server" DataSourceID="SqlDataSource3"
                                    DataTextField="name"
DataValueField="sp_id">
                                </asp:DropDownList>
                                <asp:RequiredFieldValidator
ID="RequiredFieldValidator2" runat="server"
ControlToValidate="ddlServiceProvider"
                                    ErrorMessage="Carrier is required."
ToolTip="Carrier is required."
ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
                                <asp:SqlDataSource ID="SqlDataSource3"
runat="server" ConnectionString="<%$ ConnectionStrings:OMSConnectionString
%>"
                                    SelectCommand="SELECT [sp_id], [name]
FROM [service_provider] ORDER BY [name]"></asp:SqlDataSource>
                            </td>
                        </tr>
                        <tr>
                            <td class="label">
                            </td>
                            <td class="textBox">
                                <asp:Button ID="Button1" runat="server"
OnClick="Button1_Click" Text="Save" ValidationGroup="CreateUserWizard1"
/></td>
                        </tr>
                        <tr>
                            <td class="label">
                            </td>
                            <td class="textBox">
                                <asp:RegularExpressionValidator
ID="RegularExpressionValidator2" runat="server"
ControlToValidate="txtAreaCode"
                                    Display="Dynamic"
ErrorMessage="Invalid area code format" ToolTip="invalid area code format"
                                    ValidationExpression="\d{3}"
ValidationGroup="CreateUserWizard1">Invalid area code
format</asp:RegularExpressionValidator>
                                <br />
                                <asp:RegularExpressionValidator
ID="RegularExpressionValidator1" runat="server"
ControlToValidate="txtMobileNumber"
                                    ErrorMessage="Invalid mobile number
format" ToolTip="Invalid mobile number format"
```

```
                                    ValidationExpression="\d{3}-?\d{4}"
ValidationGroup="CreateUserWizard1">Invalid mobile number
format</asp:RegularExpressionValidator><br />
                                <asp:RegularExpressionValidator
ID="RegularExpressionValidator3" runat="server"
ControlToValidate="txtMsgPerDay"
                                    ErrorMessage="Messages per day must be
a number" ToolTip="Messages per day must be a number"
                                    ValidationExpression="\d+"
ValidationGroup="CreateUserWizard1">Messages per day must be a
number</asp:RegularExpressionValidator><br />
                                <asp:RegularExpressionValidator
ID="EmailExp" runat="server" ControlToValidate="txtEmail"
                                    ErrorMessage="Invalid e-mail format."
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"

ValidationGroup="CreateUserWizard1">Invalid e-mail
format.</asp:RegularExpressionValidator><br />
                                <asp:CompareValidator ID="PasswordCompare"
runat="server" ControlToCompare="txtPassword"
                                    ControlToValidate="ConfirmPassword"
Display="Dynamic" ErrorMessage="The Password and Confirmation Password
must match."

ValidationGroup="CreateUserWizard1"></asp:CompareValidator></td>
                        </tr>
                    </table>
                    </div>
                </asp:Panel>
              </td>
             </tr>
            </table>



    </div>
    </form>
</body>
</html>
```

Login.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
```

```
using oms_lib;

namespace signup
{
    public partial class Login : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Login1_Authenticate(object sender,
AuthenticateEventArgs e)
        {
            Boolean Authenticated;
            try
            {
                Authenticated =
Utility.isValidUser(Login1.UserName.ToString(),
Login1.Password.ToString());
                if (!Authenticated)
                {
                    Login1.InstructionText = "Invalid Username/Password";
                    //Login1.InstructionTextStyle.ForeColor =
System.Drawing.Color.RosyBrown;
                }

                oms_lib.OMSUser ou = new
oms_lib.OMSUser(Login1.UserName.ToString(), Login1.Password.ToString());

                Session.Add("ou_id", ou.getU_ID());
                Session.Add("ou", ou);
                e.Authenticated = Authenticated;
            }
            catch (Exception)
            {
                e.Authenticated = false;
            }
        }

    }
}
```

Login.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Login.aspx.cs"
Inherits="signup.Login" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <img src="OMS2.gif" alt="OMS" />
    <div>
        <asp:Login ID="Login1" runat="server" BackColor="#F7F6F3"
BorderColor="#E6E2D8" BorderPadding="4"
            BorderStyle="Solid" BorderWidth="1px" CreateUserText="Create
Account" CreateUserUrl="create.aspx"
            DestinationPageUrl="default.aspx" DisplayRememberMe="False"
Font-Names="Verdana"
            Font-Size="0.8em" ForeColor="#333333"
OnAuthenticate="Login1_Authenticate">
            <TitleTextStyle BackColor="#5D7B9D" Font-Bold="True" Font-
Size="0.9em" ForeColor="White" />
            <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
            <TextBoxStyle Font-Size="0.8em" />
            <LoginButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC"
BorderStyle="Solid" BorderWidth="1px"
                Font-Names="Verdana" Font-Size="0.8em" ForeColor="#284775"
/>
        </asp:Login>

    </div>
    </form>
</body>
</html>
```

Web.config

```
<?xml version="1.0"?>

<configuration>

    <appSettings/>
    <connectionStrings>
        <add name="OMSConnectionString" connectionString="Data
Source=msdev;Initial Catalog=OMS;Persist Security Info=True;User
ID=oms;Password=oms"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
  <location path="create.aspx">
    <system.web>
      <authorization>
        <allow users="*"/>
      </authorization>
    </system.web>
  </location>

  <system.web>
```

```
        <compilation debug="true" />

      <authentication mode="Forms">
        <forms loginUrl="Login.aspx" protection="All" timeout="65"/>
      </authentication>
      <authorization>
        <deny users="?"/>
        <allow users="*"/>
      </authorization>

    </system.web>
</configuration>
```