EE M2MConnect User Guide

Version 5.5

01/09/2015

THE UK'S BIGGEST
AND FASTEST
NETWORK

1 INTRODUCTION

1.1 What is M2MConnect?

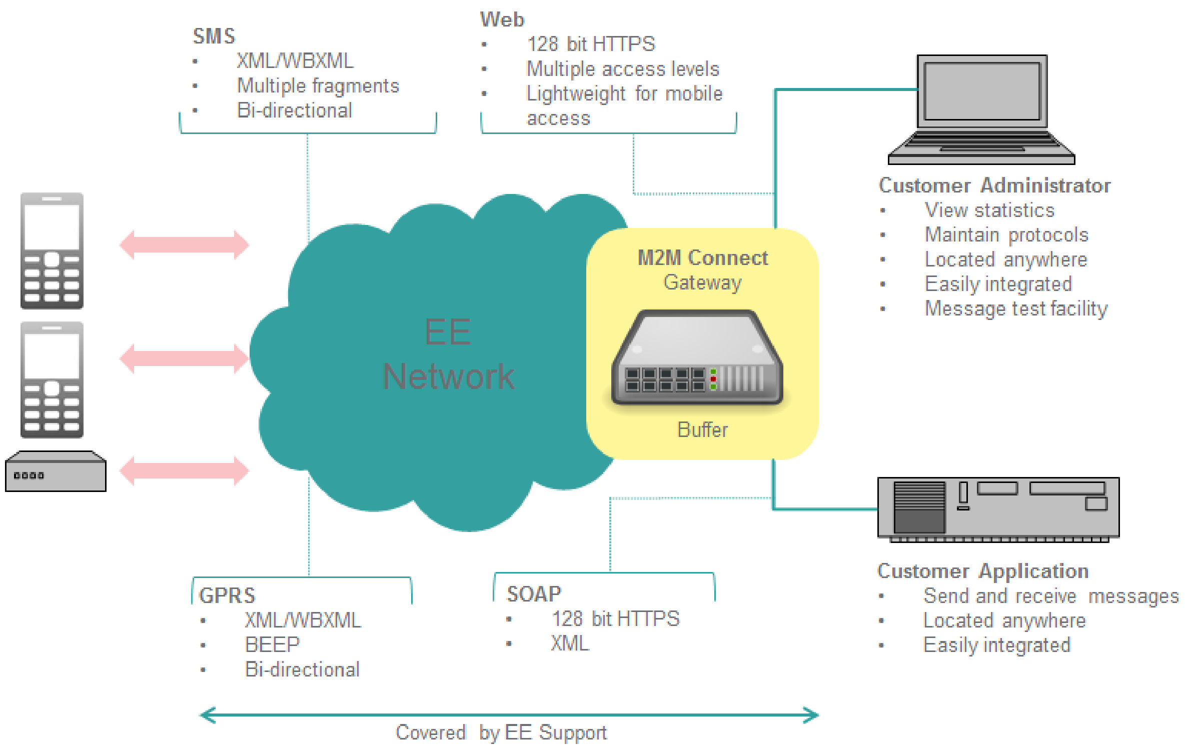
M2MConnect is a middleware gateway designed to facilitate the rapid development and deployment of robust machine to machine solutions.

The gateway is hosted and managed by EE. It acts as a data bridge between the EE GSM/GPRS network and the Internet. By using a simple interface, and connecting over the Internet, you can greatly reduce the cost and complexity of your M2M solutions.

In telemetry, or machine-to-machine (M2M) solutions, you need a method of transferring data between remote locations. By combining GSM/GPRS with the Internet you gain true remote access. In addition, M2MConnect is designed to allow you to focus on the data being transferred, rather than the idiosyncrasies of reliably transferring it.

1.2 How does it work?

In Figure 1, below, you see the typical layout of an M2MConnect installation.



Along the left hand side of the diagram are the devices. These devices can be many different types and versions. They can also use either SMS or GPRS bearers to access the GSM network. Devices are the equipment that is being monitored and/or controlled.

These devices transfer data via M2MConnect into your organisation. The contents of the data are defined by your requirements. In addition, by using XML[1], you gain flexibility for the future. XML Tags can be added at any time, and with a well designed solution, will continue to provide backwards compatibility.

Within your office environment your application receives the data from M2MConnect and presents it in a way that is useful. This may include integrating into systems like SAP, or may simply be producing Excel spreadsheets and graphs. Because M2MConnect de-couples the application from the devices, you can change the application without needing to change the devices.

In addition, M2MConnect has a secure web interface that allows you to quickly troubleshoot and diagnose any performance issues. During the development phases, this interface will allow you to test your devices and confirm that you are receiving exactly what you expect.

M2MConnect is designed to scale with your solution; therefore you continue to use the same interfaces and configuration details as you move from development into a live deployment.

1.3 SMS or GPRS?

One of the hardest parts of designing a telemetry solution is deciding on what transport should be used. This decision depends on the level of performance required, the amount of data being transferred, and possibly even the radio access devices being used.

M2MConnect gives you the flexibility to switch between GSM and GPRS transparently. Changing the bearer has no impact on your application, which means you can deploy devices now using SMS or GPRS. It also allows you to have multiple types of devices all connected to the same application, without needing to make major allowances for them within the application.

Any messages received from a SMS device will appear to the application in exactly the same fashion, using the same interfaces and format as from a GPRS device. When an application sends a message it uses a simple enumeration to select the bearer, and M2MConnect takes care of all the GSM differences.

Currently Circuit Switched Data (CSD) is not available. For most M2M applications GPRS works out as a far more cost-effective option.

1.4 Applications

Applications interact with M2MConnect to receive and send data to devices. This uses a straightforward Internet interface. Using Simple Object Access Protocol[2] (SOAP), and a secure Internet connection an application can send and receive messages.

The API consists of the following functions:

Function	Description
sendMessage	Sends a message from the Application to a device
SendMessageWithValidityPeriod	Sends a message or a binary message with a specified validity duration
sendBinarySmsMessage	Sends a binary message from the Application to a device
waitForMessage	Waits for messages from devices.
sendAndWait	Sends a message to the device, and waits for a

[1] eXtensible Markup Language. See www.w3.org/TR/xml for more information. XML is similar in structure to HTML, but is used to represent data rather than web presentation.

[2] Supported by many languages, Python, Perl, VB, .Net, C, C++, Java, etc.

	response
sendBinarySmsAndWait	Sends a binary message to the device and waits for a response
readMessages	Reads any messages that have been sent from a device
peekMessages	Reads waiting messages, but leaves them on the M2MConnect gateway
flushMessages	Deletes all waiting messages without reading them
getDeliveryReports	Gets a list of all the delivery reports
getDeliveryReportsFromDate	Gets a list of delivery reports excluding those prior to the specified date

Using WSDL[3], all you need to do is call the method and you can send and received M2M data:

```
my $soap = SOAP::Lite –>service($wsdl);


my @result = @{$soap-readMessages($username,$password,10,"") };
```

This greatly simplifies using a high performance link into the EE networks.

1.5 Concepts and Terms

At the core of M2MConnect is the *Gateway MSISDN*. This is the phone number that your devices use to send messages to your application. For each country that your solution is deployed in you will have a different Gateway MSISDN. Messages sent from M2MConnect to the device will be sent from the Gateway MSISDN.

MO and *MT* describe which direction traffic is flowing. They stand for *Mobile Originated* and *Mobile Terminated*. For M2MConnect the mobile is the device, so MO means data sent from the device to the application. MT corresponds to data sent from the application to the device.

A *device* is remote equipment that communicates with your back office system or *application*. Devices are designed and built by a device integrator, or a device manufacturer. Inside the device is a *GSM module* that actually talks to the GSM network. Some GSM modules have advanced features and may also act as a device. The *Device MSISDN* or *Terminal MSISDN* correspond to the phone number for the device. All communication with the device is done using this phone number.

An *Application* is the software running in your office or data centre that processes the returned information. It may be as simple as importing data into a spreadsheet, or as complex as integrating with an existing expense system or even SAP. Applications can also communicate with devices by sending messages.

All data is transferred as a *message*. Messages can be plain text, XML, compressed XML, or binary. All messages are discreet, allowing devices to confirm receipt, and statistics to be displayed.

Dates and times used by M2MConnect are always in UTC. This ensures that regardless of where your devices are located you will always consistently use the same time system. If

[3] Web Services Definition Language. Used by many of the SOAP libraries to provide a object that automatically deals with all the SOAP communication.

your application needs to be display in local times you can convert them using a time function or library.

1.6 Reading this Manual

The following manual is organised into sections and is structured to simplify development of a complete solution. The first section describes common administration tasks performed on the web interface. This is followed by a section on using the SOAP interfaces to develop an application. Finally there is a lot of information on how to develop a device to communicate with M2MConnect.

1.7 Security

The following functionality is temporarily suspended pending a review of customer requirements:

Security measures have been implemented to confirm that messages to and from customer devices will only be transmitted by the owner customer. Message transport cannot be initiated between an EE M2MConnect customer and the devices of a different M2MConnect customer.

This is achieved by using the first message from a device to a customer account to associate that deviceMSISDN with the customer’s account.

It is possible that due to an error in initial setup, when the first message is sent, a customer may cause his device to be associated with a different customer. In this case the EE Business Customer Services team will be able to use the M2MConnect Web Interface to remove such a faulty association, and the customer will be able to re-initialise his device to associate it with his own account number.

2 USING THE WEB INTERFACE

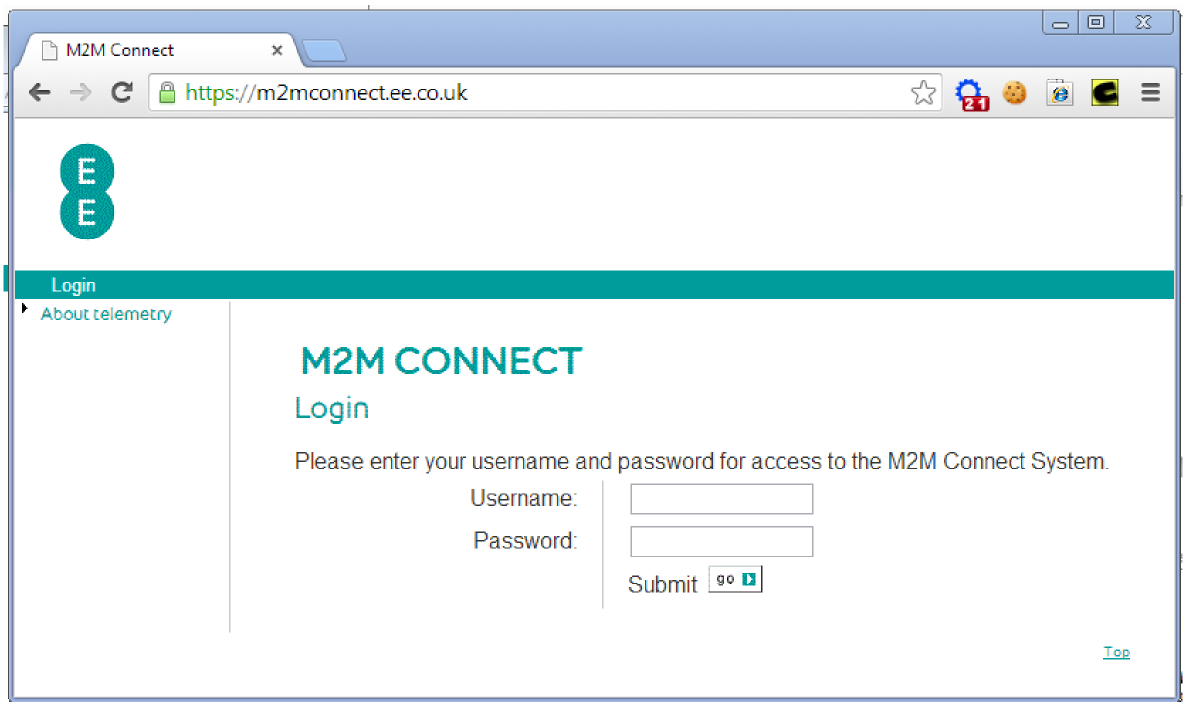
The web interface is the primary management and troubleshooting interface. It allows you to configure protocols, manage user access, view message statistics, and to send and receive messages.

2.1 Step 1: login

The first step to using EE M2MConnect is to log in to the web site. This can be accessed using an Internet browser[4], and going to the following address:

```
https://m2mconnect.ee.co.uk/
```

You should see the following screen:



M2M CONNECT

Login

Please enter your username and password for access to the M2M Connect System.

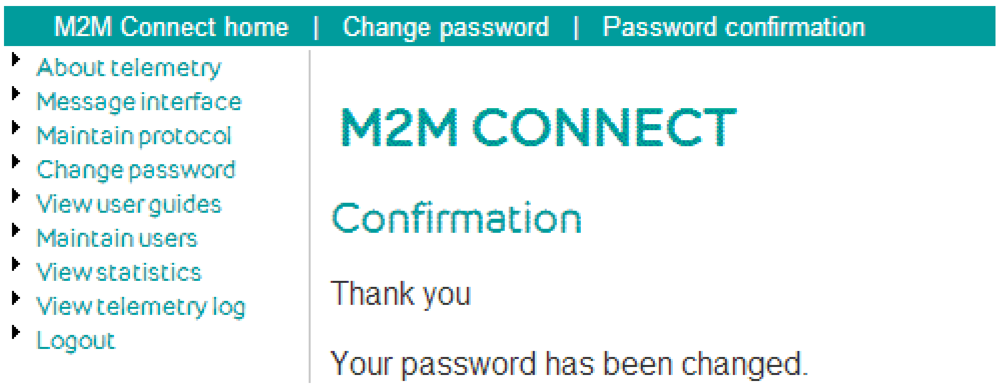
It is important to ensure that you use **https://** rather than http://. EE M2MConnect is a secure service, and is not accessible using an insecure Internet connection. The padlock symbol confirms the connection is secure. Double clicking on it will allow you to verify the security certificate, which should be signed by Verisign.

The first step for a new account is to change the generated password for a password you can remember. You initial login will have been provided to you by Business Customer Services or your account manager. It will have been configured according to the Customer Connection form. Initially your M2MConnect account will only have one user – the administrator.

1. Enter the username and password as provided to you. This password will be eight lowercase letters, which you will be required to change the first time you log in.
2. Press the submit button
3. The next screen will prompt you to enter a new password

[4] EE recommends using MS IE 8.0 or later

4. Think up a new password. This new password must be at least 8 characters long. It must also have at least one uppercase and one lowercase letter. Finally it should have one number or symbol. It is recommended that you pick a secure password, ideally not based on a word, name, birthday, or anything else that is easy to guess. For a more secure password use a combination of case, and more than one non-consecutive number or symbol. Also, your password must not be similar to your username.
5. Enter this password under “new password”, and then re-enter it under “confirm password”. Remember that passwords are case sensitive.
6. Press “go” to change your password
7. Confirmation that it has been changed will be displayed, along with the left menu bar of the M2MConnect web site.



M2M CONNECT

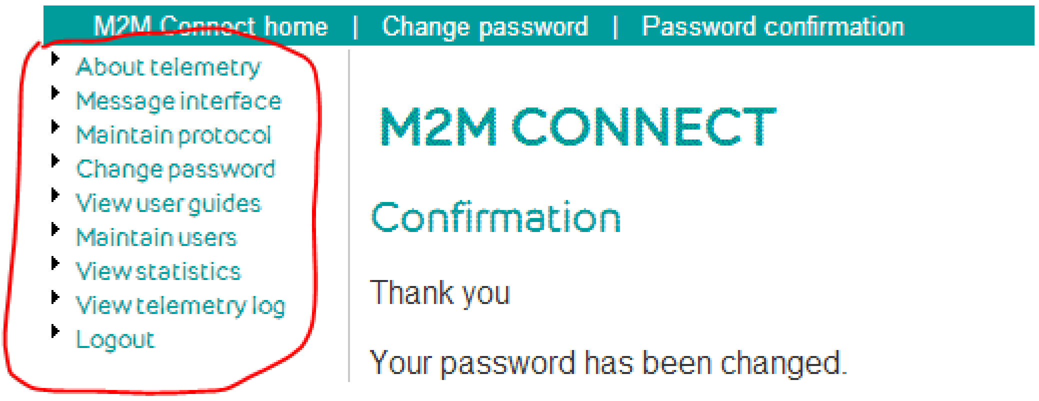
Confirmation

Thank you

Your password has been changed.

2.2 Step 2: Navigating

Once you have successfully logged in you will be presented with a control menu. This is located along the left side of the browser screen. This menu allows you to access all the features of M2MConnect:



M2M CONNECT

Confirmation

Thank you

Your password has been changed.

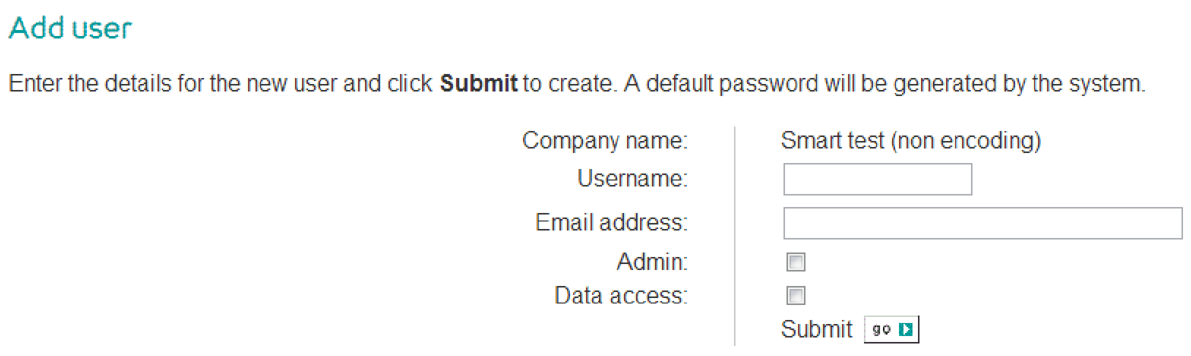
A quick summary of each of the top level features:

▸about telemetry	Takes you to the EE.co.uk homepage for M2MConnect
▸maintain users	Allows you to configure account users, including adding new users, changing users, resetting passwords, and disabling users
▸maintain protocol	Technical configuration of the account. This lists the account phone number (MSISDN), and allows you to adjust the encoding methods: no encoding or WBXML or binary
▸change password	Change the password for the currently logged in user. You can reset other users passwords in "▸maintain users"
▸view statistics	View the performance and usage statistics. This allows you to see the message traffic for all devices, as well as summary reports
▸view user guides	An electronic copy of the M2MConnect user guides
▸view telemetry log	Views the event log, which contains information about any errors and events that have occurred. This is used mostly for debugging
▸message interface	The message interface is a web based version of the SOAP interface. It allows you to simulate an application, sending messages to & receiving messages from devices
▸logout	Logs back out of M2MConnect, returning you to the login screen

The first step will be to add a normal user. This is the user you will use for day to day testing and development.

2.3 Step 3: Adding Users

New user accounts can be added through the ▸**maintain users** section. To navigate to this section, click on the maintain users option located on the main menu. In this section select the go button next to Add User at the foot of the table page. The Add New User page will be displayed, containing the following:


Add user

Enter the details for the new user and click **Submit** to create. A default password will be generated by the system.

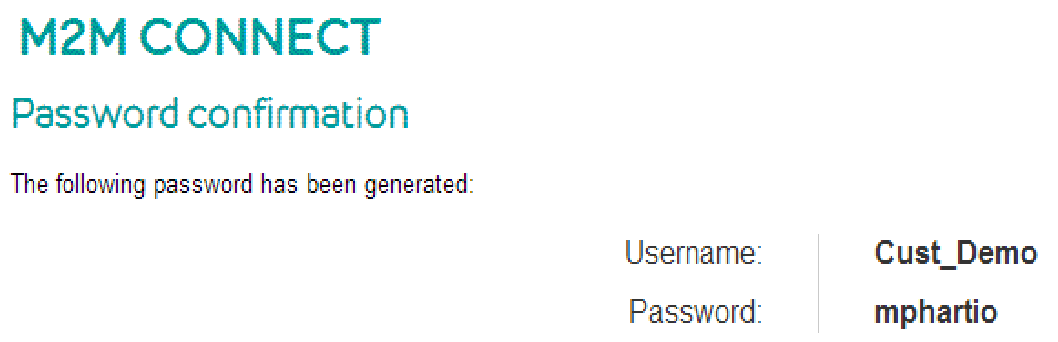
Company name:	Smart test (non encoding)
Username:	
Email address:	
Admin:	☐
Data access:	☐
	Submit go

For normal use you should add at least one user account. This account should be used by the SOAP interface to send and receive messages.

1. Enter the username for the new account. This must be at least 8 alphanumeric characters long and unique across the gateway. Once the user has been created, it isn't possible to change the username.

2. Next step is to enter the user's email address in this field. If the user is to have data access (for example, for the SOAP interface) you may leave this field blank. Not entering an email address will stop the user from receiving emails from the system relating to Quota Alerts and System Availability.
3. Check this 'Admin' select box if you want the user to have full administrator rights for the account. SOAP data users would not normally have this checked as it means the account will be able to create new users, and change protocol configurations. This box is unchecked by default.
4. To stop a user sending and receiving messages you should uncheck the 'Data Access' box This box is checked by default, and needs to be checked for SOAP message access.

After filling all the information, click the go button located at the bottom of the page. If everything has been entered correctly, the Password Confirmation page is displayed for this new user, showing the username and the auto-generated password.


M2M CONNECT

Password confirmation

The following password has been generated:

Username:	Cust_Demo
Password:	mphartio

This password should be used by the user for the first login, and will have to be changed at that point.

Once a user has been created you can enable and disable them through ▸**maintain users**. Each user can be in one of the following three states:

State	Description
Enabled	Normal, active user
Locked	A user who has entered their password incorrectly too many times. This user can be unlocked by changing the state to Enabled. Alternatively, if they have forgotten their password, you can click the 'Change Password' button to generate a new one.
Disabled	A user who is no longer allowed to access the system
Change Password	A user who has had a new password generated, but hasn't logged in to change it yet. You can disable this account, or select 'Change Password' to generate a new password.

2.4 Step 4: Sending a message

The message interface acts as a virtual application, allowing you to manually interact with devices. This interface can be used to confirm messages received, looking at the contents before they are passed on to the application. It also allows you to send test messages to devices.

The first thing to try is sending an SMS to your EE M2MConnect SIM Card[5]. For the purposes of this exercise, you should have the SIM Card in a normal phone.

1. Click on ►**message interface** in the left menu, and then select ►**send message**
2. Type in the full phone number under **terminal MSISDN** for your EE SIM Card. This must be in international format, using +, for example +447968429639. Also, it should not include spaces.
3. If the **delivery reports** check box is displayed it may optionally be checked. This will return confirmation of whether or not the device received the message.
4. Under the **bearer** option select “SMS”. If this isn’t displayed you can contact customer services to request that SMS is added to your account. Alternatively, you could use the GPRS test device to receive messages.
5. Enter some text into the **message** box and click on send to send it to your phone.

The SMS message should be received by your phone shortly. It will have been sent from your M2MConnect Gateway MSISDN, which is the number assigned to your account. If you are an international customer you will have multiple Gateway MSISDNS, one for each country you operate in.

2.5 Step 5: Receiving a message

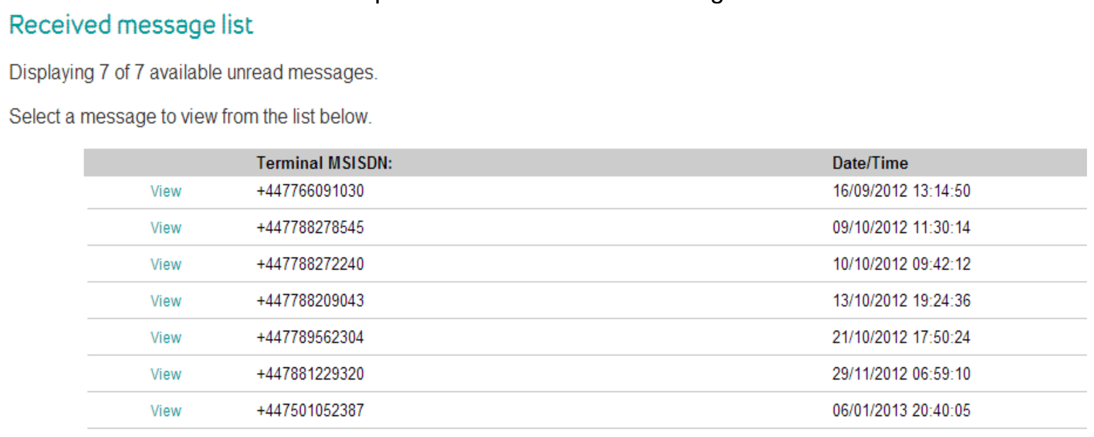
Viewing messages that have been received by the gateway is very easy. By default this will leave the messages available for your application, allowing you to use it during troubleshooting.

1. The first step is to send an SMS message to your gateway MSISDN. This is the phone number that you received the message from in Step 4[6]. For our sample you can send any information in the message.
2. On M2MConnect click on ►**message interface**. This will take you straight into the first screen for reading messages.
3. First, you may specify the maximum number of messages to return (the default is 100 messages at a time). Messages are returned oldest first.
4. The **Terminal MSISDN** allows you to look for messages from a specific device (or terminal). If you leave it blank then all messages can be returned.
5. Selecting **Mark as Read** will remove messages from M2MConnect after they’ve been read. By default this isn’t checked, which allows you to take a ‘peek’ at what is there.
6. Finally, if your company is operating in more than one country you can select which countries to return messages from. By default it will return messages from all countries.
7. Click on **submit**.
8. Another screen will be displayed listing all of the messages that are currently buffered in the gateway, (subject to the number of messages requested at a time).

[5] M2MConnect can only communicate over GPRS with EE SIM Cards.

[6] You can also locate the number under ►**maintain protocol**

The Date/Time is in GMT and represents the time the message was received.



Received message list

Displaying 7 of 7 available unread messages.

Select a message to view from the list below.

	Terminal MSISDN:	Date/Time
View	+447766091030	16/09/2012 13:14:50
View	+447788278545	09/10/2012 11:30:14
View	+447788272240	10/10/2012 09:42:12
View	+447788209043	13/10/2012 19:24:36
View	+447789562304	21/10/2012 17:50:24
View	+447881229320	29/11/2012 06:59:10
View	+447501052387	06/01/2013 20:40:05

9. Clicking on **view** will bring up the contents of each message. This is wrapped in an XML header

```
<messagerx>
  <sourcemsisdn>447966538230</sourcemsisdn>
  <destinationmsisdn>447817814211</destinationmsisdn>
  <receivedtime>23/03/2004 09:35:31</receivedtime>
  <bearer>SMS</bearer>
  <messageref>0</messageref>
  <message>Reply received</message>
</messagerx>
```

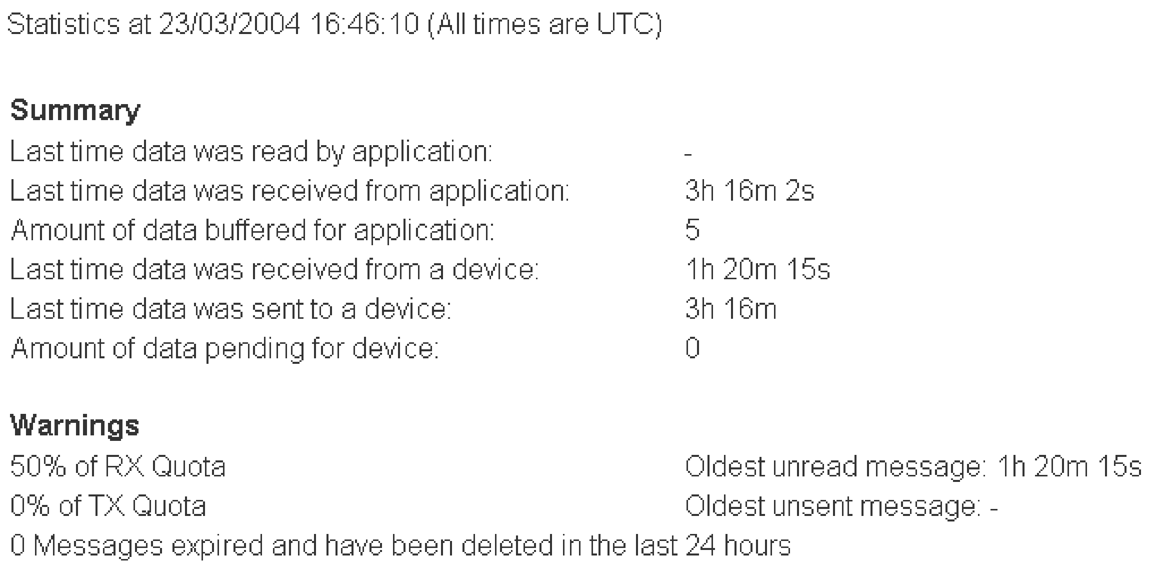
10. In the data displayed is a set of tags. Although these are described in detail in the SOAP section, the most interesting fields are as follows:
11. The **sourcemsisdn** corresponds to the device that sent the message. The sourcemsisdn is always provided in the message, even if the device is communicating using GPRS
12. The **bearer** is how the message was received by M2MConnect. It will contain either SMS or GPRS.
13. The **message** contains the data of the message sent from the device. If the message has been sent in binary mode, it will be displayed in "Base 64" format.

In addition to listing the messages already buffered on M2MConnect you can wait for a message to be received by using ▸**message interface** ▸**wait for messages**. This will wait for a message to be displayed, and then display it on the screen. Reading a message using wait for message will mark it read, and remove it from the gateway.

2.6 Step 6: Viewing Statistics

Statistics provides an interface to view the data traffic and can be helpful in troubleshooting common problems by providing a summary report of message volumes, for each device, over a period of time. This section can be accessed by navigating to ▸**view statistics** option in main menu.

1. Click on ▸**view statistics** to see summary statistics for your account.



Statistics at 23/03/2004 16:46:10 (All times are UTC)

Summary

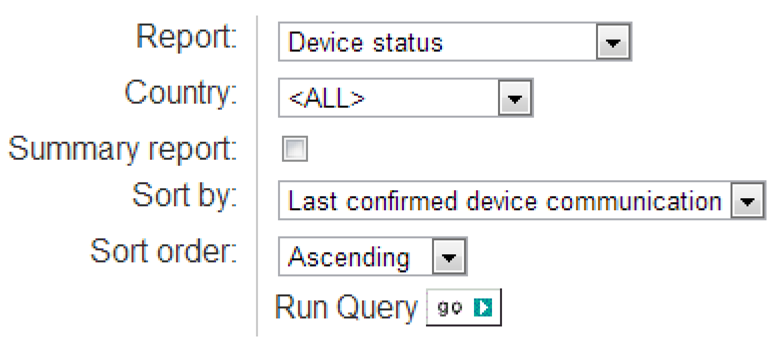
Last time data was read by application:	-
Last time data was received from application:	3h 16m 2s
Amount of data buffered for application:	5
Last time data was received from a device:	1h 20m 15s
Last time data was sent to a device:	3h 16m
Amount of data pending for device:	0

Warnings

50% of RX Quota	Oldest unread message: 1h 20m 15s
0% of TX Quota	Oldest unsent message: -

0 Messages expired and have been deleted in the last 24 hours

2. In the summary section you can see basic information on the last time messages were sent and received. The **application** figures refer to data being transferred over the SOAP interface. In this example there are 5 messages buffered for the application to fetch over SOAP.
3. The **device** figures correspond to communications with the device. So 3 hours and 16 minutes ago data was sent to a device by M2MConnect. GPRS messages are held on M2MConnect until the device fetches them, and will be listed as **pending**.
4. Under warnings you get a quick overview of message quotas. The **RX Quota** is the number of messages that you can have buffered for an application. These are messages sent from the device (MO). If your application isn't reading messages your quota will gradually fill up, eventually running the risk of discarding messages when it is exceeded[7].
5. The **TX Quota** behaves in the same fashion, but is for messages waiting to be sent to devices.
6. Messages can **expire** in two ways; automatically when they are older than 10 days, or when you exceed your quota. Messages buffered for either the application or the device can expire.
7. From the main statistics page it is possible to select from three different reports by using the query form at the bottom.



Report:	Device status
Country:	<ALL>
Summary report:	☐
Sort by:	Last confirmed device communication
Sort order:	Ascending
	Run Query go

[7] Quotas are set per customer and can be adjusted if you need larger storage. Please contact your account manager.

Report

This is a drop down select box. You can select the statistics report based on three options: Device Status, Message Counts, and Device Status MSISDN. Each report will have a different combination of the following fields available.

Country

Choose this option if you want to view country specific report. This option applies if you have subscribed for M2MConnect in more than one country. The default value of this field is ALL countries.

Start Date

Start Date restricts the report to start from the specified date and time. This field is in the format dd/mm/yyyy hh:mm:ss, and must be specified in UTC. If the time component is not entered then the entire day will be included. If this field is left blank the report will include all records from the earliest date possible.

End Date

End Date stops reporting after the specified date and time. This field is in the format dd/mm/yyyy hh:mm:ss and must be specified in UTC. If the time component is not entered then the entire day will be included. If the date / time is left blank the report will include all records up to the current time.

Specific MSISDN

If you wish to view the statistics relevant to a specific device, enter the MSISDN of the device in this field.

Summary Report

Check this box if you want the statistics to be summarized into one line. Summary reports can not be sorted as they only contain one line of information.

Sort By

If you wish to sort the record according to a specific column, select the desired column header from this select box. The options in this select box will depend on the type of report you have selected in the Report field.

Sort Order

The sort order is which order you wish the sort key to be sorted on. By default it is ascending, so greatest values first.

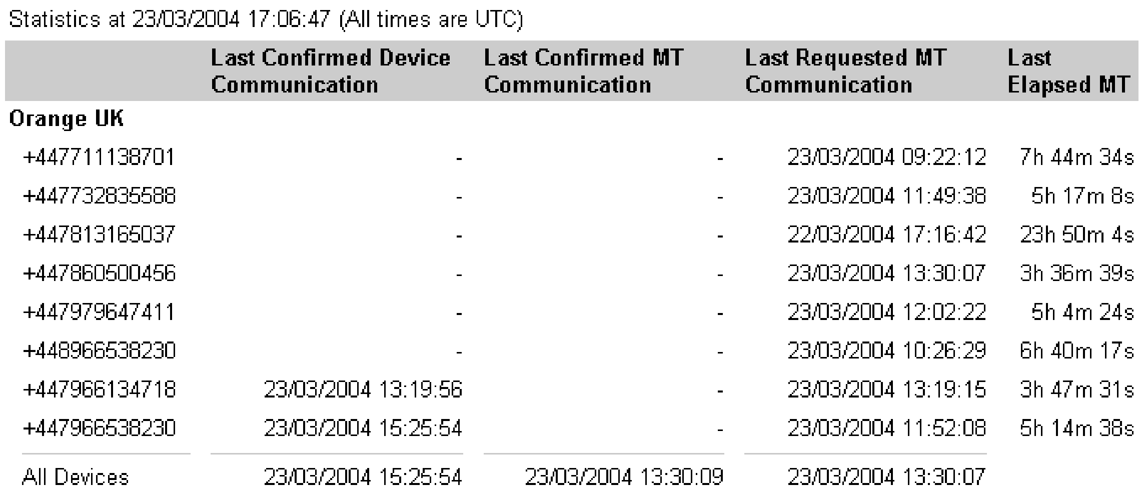
8. If you run the **Message Count** Report

The Message Count report provides a count of the number of messages that were sent. If you run the report without selecting the End Date it will also detail any messages that are currently pending.

There are three columns under each category, MO and MT. These columns show the total number of messages, number of messages that are pending on the Gateway and the number of messages that were not delivered to the destination.

The MT header represents those messages that were sent to the device through the gateway and the MO refers to the messages received from the device.

9. If you run the default **device status** report you will see the following screen:


Statistics at 23/03/2004 17:06:47 (All times are UTC)

	Last Confirmed Device Communication	Last Confirmed MT Communication	Last Requested MT Communication	Last Elapsed MT
Orange UK				
+447711138701	-	-	23/03/2004 09:22:12	7h 44m 34s
+447732835588	-	-	23/03/2004 11:49:38	5h 17m 8s
+447813165037	-	-	22/03/2004 17:16:42	23h 50m 4s
+447860500456	-	-	23/03/2004 13:30:07	3h 36m 39s
+447979647411	-	-	23/03/2004 12:02:22	5h 4m 24s
+448966538230	-	-	23/03/2004 10:26:29	6h 40m 17s
+447966134718	23/03/2004 13:19:56	-	23/03/2004 13:19:15	3h 47m 31s
+447966538230	23/03/2004 15:25:54	-	23/03/2004 11:52:08	5h 14m 38s
All Devices	23/03/2004 15:25:54	23/03/2004 13:30:09	23/03/2004 13:30:07	

The Device Status Report displays the last communication times for all active devices. It allows you to quickly spot which devices have not communicated recently.

MSISDN

The left hand column shows the country the MSISDN communicated with, as well as the MSISDN of the device. Clicking on the MSISDN will take you to the Device Status by MSISDN report.

Last Confirmed Device Communication

This column shows the time when the device last communicated with M2MConnect. This is the last time that a device either established a GPRS BEEP session, M2MConnect received an MT SMS delivery report, or M2MConnect received an MO SMS.

Last Confirmed MT Communication

This column indicates the time when M2MConnect last received a MT delivery report or the device requested a message download through BEEP. This will not be available if delivery reports were not selected when the message was sent.

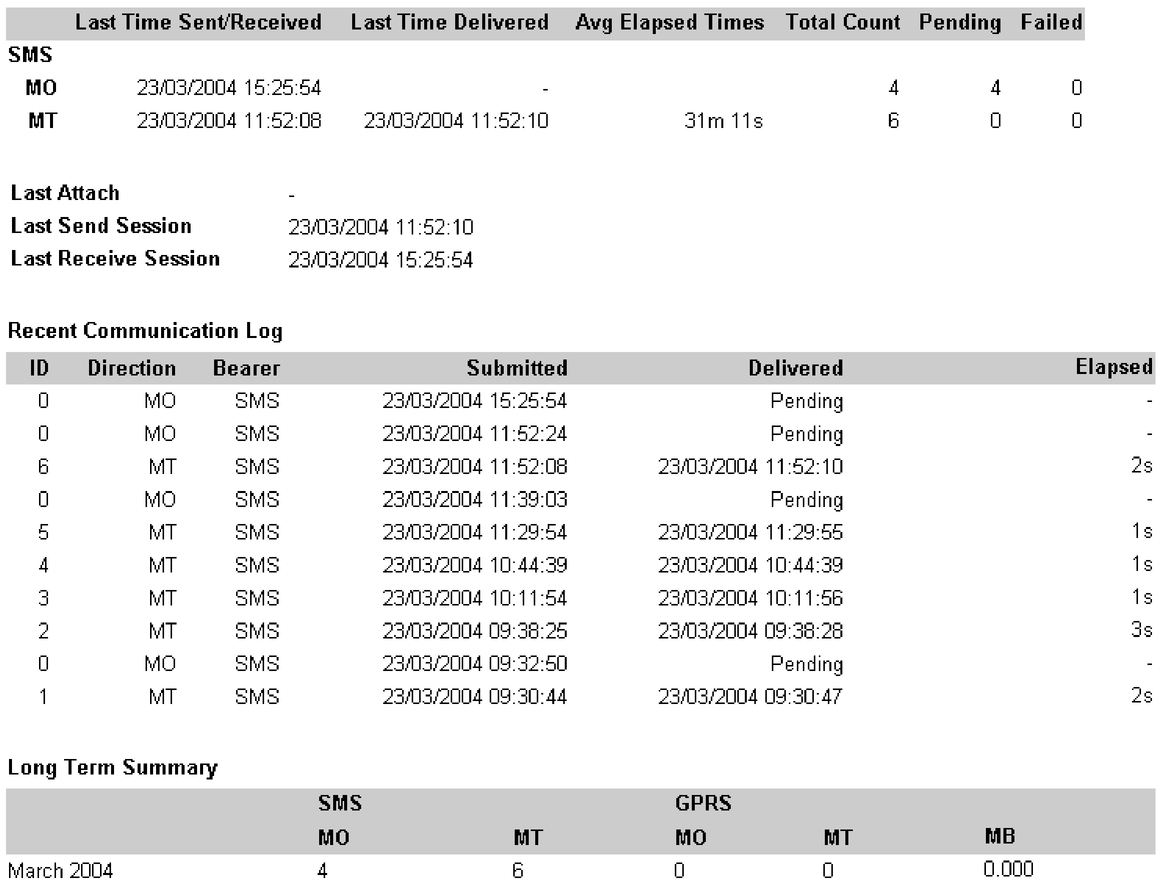
Last Requested MT Communication

This column shows the last time an application sent a message to the device.

Last Elapsed MT

This column shows the total time taken for the last MT communication with the device. This will not be available if delivery reports were not selected when the message was sent.

10. If you click on the device MSISDN it will take you into the **detailed summary** of the device’s communication.



	Last Time Sent/Received	Last Time Delivered	Avg Elapsed Times	Total Count	Pending	Failed
SMS						
MO	23/03/2004 15:25:54	-		4	4	0
MT	23/03/2004 11:52:08	23/03/2004 11:52:10	31m 11s	6	0	0

Last Attach	-
Last Send Session	23/03/2004 11:52:10
Last Receive Session	23/03/2004 15:25:54

Recent Communication Log

ID	Direction	Bearer	Submitted	Delivered	Elapsed
0	MO	SMS	23/03/2004 15:25:54	Pending	-
0	MO	SMS	23/03/2004 11:52:24	Pending	-
6	MT	SMS	23/03/2004 11:52:08	23/03/2004 11:52:10	2s
0	MO	SMS	23/03/2004 11:39:03	Pending	-
5	MT	SMS	23/03/2004 11:29:54	23/03/2004 11:29:55	1s
4	MT	SMS	23/03/2004 10:44:39	23/03/2004 10:44:39	1s
3	MT	SMS	23/03/2004 10:11:54	23/03/2004 10:11:56	1s
2	MT	SMS	23/03/2004 09:38:25	23/03/2004 09:38:28	3s
0	MO	SMS	23/03/2004 09:32:50	Pending	-
1	MT	SMS	23/03/2004 09:30:44	23/03/2004 09:30:47	2s

Long Term Summary

	SMS		GPRS		
	MO	MT	MO	MT	MB
March 2004	4	6	0	0	0.000

If you have the Device Status by MSISDN Report full details of all recent communication with the device will be listed. This report is split into three parts covering latests statistics, a recent communications log, and a historical record.

Latest Statistics

The latest statistics section contains information from the recent transaction log. It doesn't cover periods greater than 10 days.

Last Time Sent/Received

Shows the time when the last message was queued for transmission:

Bearer	Direction	Description
SMS		
	MO	The time the message was received by M2MConnect
	MT	The last time a message was sent from the application
GPRS		
	MO	This is the time the device received the OK message over BEEP when submitting the msg
	MT	The last time a message was sent from the application

Last Time Delivered

Shows the time when the last message was delivered to the intended recipient:

Bearer	Direction	Description

SMS		
	MO	The last time a message was delivered to the application
	MT	If delivery reports are enabled this is the time from the delivery report. Otherwise it is ‘-‘
GPRS		
	MO	The last time a message was delivered to the application
	MT	Last time the device returned OK over the BEEP interface

Avg Elapsed Times

The average times taken for the last communication with the device. This may not be available if delivery reports were not used. Also, remember that, for SMS, a delivery report is a separate SMS that may in itself take time to be delivered.

Total Count

The total number of messages transferred through the Gateway in

Pending

The total number of messages that are waiting to be delivered to the intended recipient. It is normal for GPRS messages to be pending until the device attaches and downloads them.

Failed

The total number of messages that were not delivered. This may be as a result of an SMS failure report, or because messages have expired.

Last Attached

This is the time when the device was last connected to the Gateway using GPRS. This is based on the establishment of a PDP context.

Last Send Session

This is the time when the device last requested a BEEP send channel creation.

Last Received Session

This is the time when the device last requested a BEEP receive channel creation.

Recent Communications Log

Following the initial information is a recent communications log. This log lists each message that has been sent or received, along with which bearer and which direction it was in.

Long Term Summary

The Long Term Summary provides high level information on at least the last three months. It also indicates how many MB of GPRS traffic passed through M2MConnect[8].

2.7 Telemetry Logs

The telemetry logs are designed to be used while troubleshooting. You may be asked to review them when making support calls.

[8] This may not correspond to the amount billed as you may be billed on traffic used to maintain and establish GPRS connections.

3 M2MConnect Solutions

3.1 Steps

Key to developing a complete solution is to look at what end result is required.

- Decide what information you wish to collect from and how you wish to control your devices

Look at how this information needs to be communicated.

- Define what messages you need to use to perform this communication.
- Determine how often communications will be needed and how you will handle situations such as missing messages.

This is the basis of your message flow. If you decide to use XML or WBXML this message flow will allow you to define the tags and the structure of each message. It will also help you to determine the operating costs of the solution. BINARY mode might also be used to send general purpose data.

Finally, look at integrating your communication into your devices and solutions. The following sections define the M2MConnect protocols for your application and device.

3.2 Successful solutions

Keeping a few things in mind will ensure that your solution is as successful as possible:

- make sure your message flow is well defined
- build around the assumption that all radio networks are inherently unreliable. Although the architecture of GSM, GPRS and M2MConnect is highly robust and resilient, any communications over the air should be considered to be unreliable.
- rely on simple, discreet, data transactions. These will recover better from transmission problems
- monitor exceptions and have all information needed to identify the problem readily available. This can be as simple as checking to make sure you have received expected information when it is expected.

3.3 Message Encoding

M2MConnect supports a variety of different message encoding formats. Selection of the appropriate one will depend on your solution.

XML and WBXML

M2MConnect is based around eXtensible Markup Language (XML). This ensures that your solutions will be as flexible as necessary, as well as ensuring that they are both forwards and backwards compatible.

XML Messages are very flexible, containing both the contents, and a description of what these contents mean. This makes the messages readable by many applications. Additionally, extending the message is simply a matter of putting more information in the message.

The following are examples of complete XML messages from real applications:

- <date>0403211200</date><Label>Test_1</Label><Temperature>013.5</Temperature><Humidity>039</Humidty><WindSpeed>000.4</WindSpeed><Pressure>0992.0</Pressure><WindChill>008.3</WindChill>
- <trip><start><lat>4916.46:N</lat><long>12311.12:W</long><GMT>12000010062003</GMT></start><finish><lat>4916.46:N</lat><long>12311.12:W</long><GMT>12000010062003</GMT></finish><tripNo>23</tripNo><mileage>123</mileage><trip>

As you can see both of these are readable. Adding further information to messages (perhaps a new version of the device) doesn’t need to change how existing messages are handled.

The downside of XML is the fact that it is fairly verbose. For many applications the extra traffic will have a minimal impact on the operating cost of the solution. However, when necessary, you can compress the XML messages using Wap Binary XML (WBXML). This provides tag and attribute compression, drastically reducing the size of the data. See section 5 for more information on WBXML.

Plain Text

M2MConnect also supports sending and receiving plain text messages. This can be useful for integrating existing devices into the M2MConnect gateway.

Binary

M2MConnect allows binary messages to be sent. This can be useful to use M2MConnect as a transparent binary messaging pipe.

When using SOAP or the web interface to send messages, messages have to be in "Base 64" format.

For Binary SMS MT messages the mandatory PDU field “Data Coding Scheme” may be selected from an M2MConnect approved list. The default is F4.

Here is an example of a binary message representing a bitmap - (represented in hex-format):

```
42 4D 66 00 00 00 00 00 00 00 3E 00 00 00 28 00 00 00 0A 00 00 00 0A 00 00 00 01 00 01
00 00 00 00 00 28 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF FF 00 FF C0 00 00 C0 C0 00 00 FF C0 00 00 F7 C0 00 00 F7 C0 00 00 D6 C0 00
00 FF C0 00 00 FF C0 00 00 FF C0 00 00 FF C0 00 00
```

This binary message encoded in Base64 looks like shown below:

Qk1mAAAAAAAAAD4AAAAoAAAACgAAAAoAAAABAAEAAAAAACgAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAP///wD/wAAAwMAAAN7AAAD/wAAA98AAAPfAAADWwAAA/8AAA
P/AAAD/wAAA

4 DEVELOPING AN APPLICATION

Applications are the office side of an M2M Solution. They are responsible for retrieving the information from the device and making it available for use. Most solutions will have different requirements for their application; ranging from as simple as a spreadsheet through to Oracle or SAP integration.

The API described below is how an application communicates with M2MConnect devices.

4.1 SOAP

Simple Object Access Protocol (SOAP) is a W3C[9] specification. According to the W3C website (http://www.w3.org/TR/SOAP/), it is:

> SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

M2MConnect uses SOAP over a secure HTTP connection (https) to provide you with simple and secure access to your M2M data.

To make it even more straightforward, M2MConnect exports a Web Services Definition Language (WSDL) file that defines all of the functions available. This WSDL allows proxy objects to provide an automatic stub. For example, in Perl, using SOAP::Lite, the core code for reading messages looks like:

```
use SOAP::Lite;

my $soap = SOAP::Lite –>service($wsdl);

my @result = @{$soap-readMessages($username,$password,10,"") };
```

For many applications, you will never actually use the SOAP layers directly. This, obviously, greatly simplifies the development of the end solution.

M2MConnect has been tested with SOAP::Lite for Perl, Microsoft SOAP Toolkit 3.0 (VB/VC,etc), Microsoft .Net, and SOAP.py for Python.

For the purposes of initially working with SOAP it’s recommended that you configure your M2MConnect account to send un-encoded SMS. This will allow you to send & receive messages using your EE mobile.

Binary messages might be sent using SOAP; just make sure messages have been converted in "Base 64" format.

4.1.1 SOAP Restrictor

The following functionality is temporarily suspended pending a review of existing customer usage. However all new applications must be developed with these restrictions in mind.

[9] World Wide Web Consortium (http://www.w3.org/)

The EE M2MConnect platform can be accessed through a number of channels, some of which are designed to be used by automated processes. If a script that executes an automated process has faulty parameters on setup it is possible that excessive calls can be made on the M2MConnect platform. This in turn may affect the service of other customers.

EE M2MConnect has addressed this specifically with regard to SOAP access by means of a SOAP Restrictor. This sets limits on the number of times any individual customer can make specific SOAP calls.

These limits are:

• readMessages	maximum 1 request per second
• peekMessages	maximum 1 request per second
• getDeliveryReports	maximum 1 request every 10 seconds
• waitForMessage	maximum 1 request every 60 seconds

Any SOAP calls that exceed these limits by any individual customer will result in an error message thrown by the SOAP transaction.

These limits are viewed as extreme. EE FT-Group would recommend that where continuous polling is required the requests frequency would usually be:

• readMessages	1 request every 5 to 10 seconds
• peekMessages	1 request every 5 to 10 seconds
• getDeliveryReports	1 request every 1 minute
• waitForMessage	1 request every 5 minutes

4.2 WSDL

All applications should be developed using the Web Services Design Language file (WSDL) located at:

https://m2mconnect.ee.co.uk/orange-soap/services/MessageServiceByCountry?wsdl

Although this isn't the most readable file, it allows the various SOAP libraries to automatically communicate with M2MConnect. Also, there are several tools available on the Internet that allow you to read and graphically display WSDL.

An older interface was available at the address below. This interface is now obsolete and is no longer formally supported.

https://m2mconnect.ee.co.uk/orange-soap/services/MessageService?wsdl

NOTE: Please do not use this older interface – this will be removed in due course

SOAP API

The following sections define the functions available using SOAP. These are the same as defined in the WSDL, and should be used as a 'quick' reference. For many of the methods there are two different signatures. These correspond to which WSDL file is used by the application, as described above.

4.2.1 sendMessage

This method sends a message from M2MConnect to a M2M device.

MessageServiceByCountry
returnCode sendMessage(username, password, deviceMSISDN, message, deliveryReport, mtBearer)

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all the SOAP requests are transferred using HTTPS this information is never sent across the network in plain text.

String deviceMSISDN

The EE phone number to send the message to. This must be international format, including the leading '+'. (eg +447968429639).

String ***message***

The plain text, XML or binary message to send to the device. Plain text messages can also be sent if the WBXML encoding is deactivated at the M2MConnect gateway. XML messages should be well formed XML including only one top level tag. Binary messages must be Base 64 encoded[10].

Because SOAP uses XML payloads you should ensure that you escape all XML indicators. These include '<', '&', and '>'. Escape characters are converted back into the original form before transmission to the device, so no extra overhead is incurred over-the-air.

Original Character	Replacement Character
<	&lt;
>	&gt;
&	&amp;

You do not need to replace the & from the replacement character with &amp;

Message to be sent	Correctly encoded
<msg/>	**&lt;**msg/**&gt;**
<msg>Hello!</msg>	**&lt;**msg**&gt;**Hello!**&lt;**/msg**&gt;**
<msg>Bob & Jane</msg>	**&lt;**msg**&gt;**Bob **&amp;** Jane**&lt;**/msg**&gt;**
<msg>Bob &amp; Jane</msg>	**&lt;**msg**&gt;**Bob **&amp;**amp; Jane**&lt;**/msg**&gt;**

Boolean ***deliveryReport***

A SOAP boolean indicating if you require a delivery report for this message or not. Delivery reports must be enabled for the M2MConnect account.

String ***mtBearer***

[10] For more information concerning Base64 encoding, please refer to RFC2045 at http://www.ietf.org.

An enumeration indicating what bearer should be used to send the message to the device. You need to ensure that the device is capable of receiving messages using the specified bearer.

Value of mtBearer	Action
“” <empty string>	Selects the default bearer for your account. If you have a GPRS account this will be GPRS, otherwise it will be SMS.
“GPRS”	Selects GPRS as the bearer. Messages will be queued until the device attaches to retrieve them. If the account doesn’t support GPRS then an error will occur.
“SMS”	Will send the message using SMS. If the account doesn’t support SMS then an error will occur.
“UGPRS”	Urgent GPRS message. This sends an SMS to the device indicating that a GPRS message is waiting for it. It behaves much like the voicemail indicator, only being sent when it is necessary.

Int ***returnCode***

The successful return from the method does not indicate that the message has reached the target device or that it has even reached the network. It merely indicates that it has been accepted into the Gateway for processing.

The returned Integer is the message reference that has been assigned to that new message. It is unique for each message transmitted to the deviceMSISDN. It can be used to identify responses from the device, particularly within the Wait For Message functions.

Error Handling

If the call fails a SOAP exception is raised. These are listed separately.

Perl SOAP::Lite example[11]

Send the message “<msg>Hello!</msg>” to 07968 429 639 using SMS. This will print the reference id on the screen.

```
' Create the SOAP object & initialise it to connect to M2MConnect
use SOAP::Lite;
my $soap = SOAP::Lite –>service($wsdl);

' Send a message to 07968429639 using SMS and without delivery reports
print $soap->sendMessage( $username, $password,
                          “07968429639”,
                          “&lt;msg&gt;Hello!&lt;/msg&gt;”,
                          SOAP::Data->type(boolean => 'false'),
                          “SMS”);
```

[11] Expects $username, $password, and $wsdl to be configured…

sendMessage has a flow control function implemented to reduce the likelihood that any particular customer can affect other customers.

If a customer sends thousands of MT messages in a short space of time he may flood the system causing slow response for other customers.

This issue is addressed by limiting how many messages a customer can send in a period of time.

The result will be a SOAP MessageServiceException “Maximum-requests-per-timeperiod threshold exceeded”, detailing the user and customer.

Any messages with this response will have been rejected and will have to be resent.

4.2.2 sendMessageWithValidityPeriod

This method sends a SMS message or a binary SMS message from M2MConnect to an M2M device. This message will expire after a specified period of non-delivery.

MessageServiceByCountry
returnCode sendMessageWithValidityPeriod(username, password, deviceMSISDN, message, deliveryReport, mtBearer, binarySmsDcs, validityDaysHoursMinutes)

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all the SOAP requests are transferred using HTTPS this information is never sent across the network in plain text.

String deviceMSISDN

The EE phone number to send the message to. This must be international format, including the leading ‘+’. (eg +447968429639).

String ***message***

The plain text, XML or binary message to send to the device. Plain text messages can also be sent if the WBXML encoding is deactivated at the M2MConnect gateway. XML messages should be well formed XML including only one top level tag. Binary messages must be Base 64 encoded[12].

Because SOAP uses XML payloads you should ensure that you escape all XML indicators. These include ‘<’, ‘&’, and ‘>’. Escape characters are converted back into the original form before transmission to the device, so no extra overhead is incurred over-the-air.

Original Character	Replacement Character
<	&lt;
>	&gt;

[12] For more information concerning Base64 encoding, please refer to RFC2045 at http://www.ietf.org.

&	&amp;

You do not need to replace the & from the replacement character with &amp;

Message to be sent	Correctly encoded
<msg/>	**&lt;**msg/**&gt;**
<msg>Hello!</msg>	**&lt;**msg**&gt;**Hello!**&lt;**/msg**&gt;**
<msg>Bob & Jane</msg>	**&lt;**msg**&gt;**Bob **&amp;** Jane**&lt;**/msg**&gt;**
<msg>Bob &amp; Jane</msg>	**&lt;**msg**&gt;**Bob **&amp;**amp; Jane**&lt;**/msg**&gt;**

Boolean ***deliveryReport***

A SOAP boolean indicating if you require a delivery report for this message or not. Delivery reports must be enabled for the M2MConnect account.

String ***mtBearer***

An enumeration indicating what bearer should be used to send the message to the device. You need to ensure that the device is capable of receiving messages using the specified bearer.

Value of mtBearer	Action
“” <empty string>	Selects the default bearer for your account. If you have a GPRS account this will be GPRS, otherwise it will be SMS.
“GPRS”	Selects GPRS as the bearer. Messages will be queued until the device attaches to retrieve them. If the account doesn’t support GPRS then an error will occur.
“SMS”	Will send the message using SMS. If the account doesn’t support SMS then an error will occur.
“UGPRS”	Urgent GPRS message. This sends an SMS to the device indicating that a GPRS message is waiting for it. It behaves much like the voicemail indicator, only being sent when it is necessary.

String ***binarySmsDcs***

This may be one of the list of approved SMS DCS values, which are currently “04” and “F4”. The default is “F4”.

String ***validityDaysHoursMinutes***

Validity Period is an optional value that specifies the number of days, hours and minutes after which an SMS message should be discarded if not delivered to the destination device. If no value is supplied the SMSC will default, typically to 7 days.

The Validity Period parameter is only available to messages sent from UK accounts. It provides a means of specifying a message validity shorter than the SMSC default. This will be useful where an application requires that a message only be delivered to a device if it can be done within a specified period eg within 2 hours.

Where specified, Validity Period must be entered as six characters in the form DDHHMM - days, hours and minutes. The maximum values are for each field are 30 days, 23 hours 59 minutes. For example a Validity Period of 2 hours would be entered as 000200.

Please be aware –UK SMSC's typically allow a maximum message validity of 7 days, but the SMSCs can be re-configured locally to override any value requested via M2MConnect.

Int ***returnCode***

The successful return from the method does not indicate that the message has reached the target device or that it has even reached the network. It merely indicates that it has been accepted into the Gateway for processing.

The returned Integer is the message reference that has been assigned to that new message. It is unique for each message transmitted to the deviceMSISDN. It can be used to identify responses from the device, particularly within the Wait For Message functions.

Error Handling

If the call fails a SOAP exception is raised. These are listed separately.

4.2.3 sendBinarySMSMessage

This method sends a Binary SMS message from M2MConnect to a M2M device.

Note! This function may only be executed for a customer account whose protocol is set to Binary.

```
MessageServiceByCountry

returnCode sendBinarySMSMessage(    username, password,
                                    deviceMSISDN,
                                    message,
                                    deliveryReport, mtBearer, binarySmsDcs)
```

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all the SOAP requests are transferred using HTTPS this information is never sent across the network in plain text.

String deviceMSISDN

The EE phone number to send the message to. This must be international format, including the leading '+'. (eg +447968429639).

String ***message***

The binary message to send to the device. Binary messages must be Base 64 encoded[13].

Because SOAP uses XML payloads you should ensure that you escape all XML indicators. These include '<', '&', and '>'. Escape characters are converted back into the original form before transmission to the device, so no extra overhead is incurred over-the-air.

Boolean deliveryReport

[13] For more information concerning Base64 encoding, please refer to RFC2045 at http://www.ietf.org.

A SOAP boolean indicating if you require a delivery report for this message or not. Delivery reports must be enabled for the M2MConnect account.

String ***mtBearer***

This must be “SMS”.

String ***binarySmsDcs***

This may be one of the list of approved SMS DCS values, which are currently “04” and “F4”. The default is “F4”.

Int ***returnCode***

The successful return from the method does not indicate that the message has reached the target device or that it has even reached the network. It merely indicates that it has been accepted into the Gateway for processing.

The returned Integer is the message reference that has been assigned to that new message. It is unique for each message transmitted to the deviceMSISDN. It can be used to identify responses from the device, particularly within the Wait For Message functions.

Error Handling

If the call fails a SOAP exception is raised. These are listed separately.

4.2.4 waitForMessage

waitForMessage is used to wait for an incoming message to be received. It is ideal for reducing message latency as the first matching received message will be returned as soon as it arrives, rather than on the polling interval.

MessageService (deprecated)
returnMsgs waitForMessage(username, password, timeout, deviceMSISDN, msgref)
MessageServiceByCountry
returnMsgs waitForMessage(username, password, timeout, deviceMSISDN, msgref, countryCode,)

This method will first check to see if there are any unread messages matching the supplied parameters. If there are the method will return after retrieving the first message as the readMessages() method.

If there are no unread messages matching the supplied parameters then the method will wait (for up to the length of the supplied timeout) for a message matching the supplied parameters to be received from a device.

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all the SOAP requests are transferred using HTTPS this information is never sent across the network in plain text.

Int ***timeout***

The number of seconds to wait for messages to be received. If no messages have been received in this time an empty array will be returned.

String ***deviceMSISDN***

The deviceMSISDN is an optional parameter and if specified (i.e. not empty) then only unread messages from the specified MSISDN will be returned. This must be specified in international '+' format.

Int ***msgref***

This is an optional filter parameter. If it is specified then only messages matching this **msgref**, and the **deviceMSISDN** will be returned. If **msgref** is specified then **deviceMSISDN** must also be specified. If you don't wish to filter on msgref set it to 0.

String ***countryCode***

When using the MessageServiceByCountry WSDL you can also filter based on the countryCode of the device. This allows you to retrieve data for each country separately. Country codes should be specified as the two or three digit prefix. For UK this is 44, for France 33, for Belgium 32, and for Romania 40.

String [] ***returnMsgs***

This method will return an array of XML messages. If no messages are available an empty array will be returned. Binary messages are encoded in "Base 64" format.

4.2.5 sendAndWait

This method allows a combined SOAP call for sendMessage and waitForMessage methods. It first sends a message to the target device, and then waits for the incoming messages to be received.

MessageServiceByCountry	
returnMsgs sendAndWait(	username, password, timeout, deviceMSISDN, message, deliveryReport, mtBearer)

This method will take the supplied message and pass it through the messaging layers and send it to the target device specified by destinationaddress.

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all SOAP requests are transferred using HTTPS, this information is never sent across the network in plain text.

Int ***timeout***

The number of seconds to wait for message to be received. If no messages have been received in this time, an empty array will be returned.

String ***deviceMSISDN***

The MSISDN number of the device that will receive the message. This number should be specified in international format.

(In this case, since the Country is determined by the destination deviceMSISDN, no countryCode is enterd.)

String ***message***

The plain text, XML or binary message to send to the device. Plain text messages can also be sent if the WBXML encoding is off at the M2MConnect gateway. XML messages should be well formed XML including only one top level tag. Binary messages must be Base 64 encoded[14].

Boolean ***deliveryReport***

A SOAP boolean indicating if you require a delivery report for this message or not. Delivery reports must be enabled for the M2MConnect account.

String ***mtBearer***

An enumeration indicating what bearer should be used to send the message to the device. You need to ensure that the device is capable of receiving messages using the specified bearer.

Value of mtBearer	Action
“” <empty string>	Selects the default bearer for your account. If you have a GPRS account this will be GPRS, otherwise it will be SMS.
“GPRS”	Selects GPRS as the bearer. Messages will be queued until the device attaches to retrieve them. If the account doesn’t support GPRS then an error will occur.
“SMS”	Will send the message using SMS. If the account doesn’t support SMS then an error will occur.
“UGPRS”	Urgent GPRS message. This sends an SMS to the device indicating that a GPRS message is waiting for it. It behaves much like the voicemail indicator, only being sent when it is necessary.

String [] ***returnMsgs***

This method will return an array of XML messages. If no messages are available, then an array containing one entry with the message id of the sent message will be returned. This can be used to call waitForMessages to wait for a longer time. Binary messages are encoded in "Base 64" format.

The following example shows how to use sendAndWait function in VB

VB 6 Example
Send message and wait for any incoming messages.
Set SOAPObj = CreateObject(“MSSOAP.SoapClient30”)

[14] For more information concerning Base64 encoding, please refer to RFC2045 at http://www.ietf.org.

```
Call SOAPObj.mssoapinit(wsdl)
‘Call the sendAndWait method
NewMessages = SOAPObj. sendAndWait(username, password, 60,
“+447968429639”, message, true, “SMS”)


For each msgs in NewMessages
InboxMsg = InboxMsg & msgs & vbcrlf
Next
```

4.2.6 sendBinarySMSAndWait

This method sends a Binary SMS message from M2MConnect to a M2M device.

Note! This function may only be executed for a customer account whose protocol is set to Binary.

```
MessageServiceByCountry

returnCode sendBinarySMSAndWait(     username, password,
                                timeout,
                                deviceMSISDN,
                                message,
                                deliveryReport, mtBearer, binarySmsDcs)
```

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all the SOAP requests are transferred using HTTPS this information is never sent across the network in plain text.

Int ***timeout***

The number of seconds to wait for message to be received. If no messages have been received in this time, an empty array will be returned.

String deviceMSISDN

The EE phone number to send the message to. This must be international format, including the leading ‘+’. (eg +447968429639).

String ***message***

The binary message to send to the device. Binary messages must be Base 64 encoded[15].

Because SOAP uses XML payloads you should ensure that you escape all XML indicators. These include ‘<’, ‘&’, and ‘>’. Escape characters are converted back into the original form before transmission to the device, so no extra overhead is incurred over-the-air.

Boolean deliveryReport

[15] For more information concerning Base64 encoding, please refer to RFC2045 at http://www.ietf.org.

A SOAP boolean indicating if you require a delivery report for this message or not. Delivery reports must be enabled for the M2MConnect account.

String ***mtBearer***

This must be “SMS”.

String ***binarySmsDcs***

This may be one of the list of approved SMS DCS values, which are currently “04” and “F4”. The default is “F4”.

String [] ***returnMsgs***

This method will return an array of XML messages. If no messages are available an empty array will be returned. Binary messages are encoded in "Base 64" format.

Error Handling

If the call fails a SOAP exception is raised. These are listed separately.

4.2.7 readMessages

This method will return an array of messages that have not yet read. Messages will only be returned once by this method.

```
MessageService (deprecated)

returnMsgs readMessages(username, password,
                        count,
                        deviceMSISDN )

MessageServiceByCountry

returnMsgs readMessages(username, password,
                        count,
                        deviceMsisdn,
                        countryCode )
```

String **username**

M2MConnect username, used to validate access to the account. The user must have DATA access.

String **password**

The M2MConnect password for the username. As all the SOAP requests are transferred using HTTPS this information is never sent across the network in plain text.

Int **count**

The maximum number of messages to return. If less than count messages are available, then all waiting messages will be returned.

String **deviceMsisdn**

The deviceMsisdn is an optional parameter and if specified (i.e. not empty) then only unread messages from the specificed MSISDN will be returned. This must be specified in international ‘+’ format.

String **countryCode**

When using the MessageServiceByCountry WSDL you can also filter based on the countryCode of the device. This allows you to retrieve data for each country separately. Country codes should be specified as the two or three digit prefix. For UK this is 44, for France 33, for Belgium 32 and for Romania 40.

String [] **returnMsgs**

This method will return an array of XML messages. If no messages are available an empty array will be returned.

Each M2M message will be wrapped in the following envelope:

Tag	Description
<**messagerx**>	Indicates a received message, contains the following tags
<sourcemsisdn> 4479684296200 **</sourcemsisdn>**	The MSISDN of the device that sent the message. This MSISDN will have been validated by the EE network. This number will be in international format, without a leading +
<destinationmsisdn> 447968429639 **</destinationmsisdn>**	The MSISDN that the message was sent to. This corresponds with your one of your M2MConnect gateway MSISDNs. It is included to ensure you can identify traffic if you merge messages from multiple M2MConnect accounts. This number will be in international format, without a leading +
<receivedtime> 17/06/2003 18:59:10 **</receivedtime>**	The date and time M2MConnect received the message. This may differ from the time the device sent the message. DD/MM/YYYY HH:MM:SS. This time will be in UTC rather than local time.
<bearer> SMS **</bearer>**	The bearer the message was received on. This will be either “GPRS” or “SMS”. This will be the same text as used by the mtBearer flag in sendMessage
<messageref> 10 **</messageref>**	The reference number assigned to the message. This can be used to match a response from an M2M device to a previous request.
<message> <vehicle><reg>RK02 FGX</reg> <mph>10</mph></vehicle> **</message>**	The contents of the message, as received from the device.
</messagerx>	Closes the top level messagerx envelope

This envelope may be extended by the addition of new tags in the future. Also, the order of the tags is not defined.

Error Handling

If the call fails a SOAP exception is raised. These are listed separately.

Perl SOAP::Lite example[16]

[16] Expects $username, $password, and $wsdl to be configured…

Read and display up to 10 messages, without filtering on source MSISDN

```
' Create the SOAP object & initialise it to connect to M2MConnect
use SOAP::Lite;
my $soap = SOAP::Lite –>service($wsdl);

' read 10 messages
my @result = @{$soap->readMessages($username,$password,10,"") };

' How many messages were returned?
my $count = $#result + 1;
print "$count messages returned\n\n";

' Print each message to the screen
if ( $count > 0 ) {
        foreach my $line (@result) {
          print "$line\n";
        };
};
```

Excel VBA example[17]

Read and insert into the current sheet up to 10 messages, without filtering on source MSISDN

```
' Create the SOAP object & initialise it to connect to M2MConnect
Set soapClient = CreateObject("MSSOAP.SoapClient30")
Call soapClient.mssoapinit(wsdl)

' Get a message from the server, for any MSISDN
Results = soapClient.readMessages(userName, password, 10, "")

' Row to start on
Count = 1

' Now add them to the spreadsheet
For Each msg In Results
```

[17] Expects userName, password, and wsdl to be configured…

```
        If IsEmpty(msg) Then
          msg = "--- No Messages ---"
       End If

        ActiveSheet.Cells(Count, 1) = msg
        Count = Count + 1

Next
```

4.2.8 peekMessages

This method also returns an array of messages that are not marked as read. The difference between peekMessages and readMessages is that unlike readMessages, the messages are not removed from the M2MConnect Gateway after the read. Thus with peekMessages, unread messages can be fetched several times.

MessageService (deprecated)
returnMsgs peekMessages(username, password, count, deviceMSISDN)
MessageServiceByCountry
returnMsgs peekMessages(username, password, count, deviceMsisdn, countryCode)

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all SOAP requests are transferred using HTTPS, this information is never sent across the network in plain text.

Int ***count***

This is the maximum number of messages to return. If less than count messages are available, all the unread messages will be returned.

String ***deviceMsisdn***

It is an optional parameter and if specified, then the unread messages from the given MSISDN will be returned. The deviceMsisdn should be specified in international format.

String [] ***returnMsgs***

This method will return an array of XML messages. If no messages are available, then an empty array will be returned. Binary messages are encoded in "Base 64" format.

The following example shows how to use peekMessages function in ASP:

ASP Example
Read and display new messages using peekMessages

```
<%
Set SOAPObj = Server.CreateObject("MSSOAP.SoapClient30")
Call SOAPObj.mssoapinit(wsdl)
'Call the peekMessages method
NewMessages = SOAPObj.peekMessages(username, password, 10, "")
Inum = 1 'Increment Operator
%>
<TABLE WIDTH="100%">
  <TR>
      <TD WIDTH="30%">Message Number</TD>
        <TD width="70%">Message</TD>
     </TR>
<% For each msgs in NewMessages %>
      <TR>
               <TD><%=Inum%></TD>
              <TD><%=msgs%></TD>
      </TR>
<%
      Inum = Inum+1
 Next
%>
</TABLE>
```

4.2.9 flushMessages

This method deletes any messages waiting for the application from the gateway. Once messages have been flushed you can not retrieve them.

MessageService WSDL (deprecated)
returnCode flushMessages(username, password, deviceMSISDN)
MessageServiceByCountry WSDL
returnCode flushMessages(username, password, deviceMSISDN, countryCode)

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all the SOAP requests are transferred using HTTPS this information is never sent across the network in plain text.

String ***deviceMSISDN***

The deviceMSISDN is an optional parameter and if specified (i.e. not empty) then only messages from the specified MSISDN will be deleted. This must be specified in international '+' format. If the deviceMSISDN is left empty then all messages will be deleted.

String ***countryCode***

When using the MessageServiceByCountry WSDL you can also filter based on the countryCode of the device. This allows you to flush messages for each country separately. Country codes should be specified as the two or three digit prefix. For UK this is 44, for France 33, for Belgium 32.

Boolean ***returnCode***

Returns true if any messages were flushed, and false if no messages were marked as flushed.

4.2.10 getDeliveryReports / getDeliveryReportsFromDate

These methods will return an array of available Delivery Reports for sent messages from the database for the MSISDN number associated with the specified company. Only messages that have been sent requesting a delivery report will receive delivery reports[18].

Delivery Report options are maintained using the web service Maintain Protocol screen - for details see the M2MConnect Customer User Guide.

Delivery Reports by Exception

Delivery Reports within M2MConnect are used to confirm to a customer that a specific message has been received by a device. Some EE M2MConnect customers send a large number of messages to devices, and are particularly interested only in those messages that cannot be confirmed to have arrived at the device. Such a customer may choose to retrieve only those Delivery Reports that have failed, or have not yet been received. The customer would set an exception time span, beyond which a lack of response from the device will be deemed to be a failure.

i.e if the exception time is set at 10 minutes, and a delivery report is not received from the device to notify of success or failure within that time, it will be assumed that delivery of the original message to the device has failed.

(Subsequent arrival of the Delivery Report from the device will update the delivery status.)

This setting is found on the “maintain protocol” page. If Delivery Reports are enabled for this account a radio button can be used to choose between “Delivery Reports” and “Exception Reports”. Selecting “Exception Reports” will reveal the Exception Time field.

Delivery reports As MO.

Reports as MO adds a new way of viewing delivery reports within the M2MConnect platform.

Within the SMS network Delivery Reports are distinct messages, the same as normal text messages. These are special purpose messages that are transmitted back to a sender to notify them that an intended SMS has arrived (or has failed to arrive) at its destination.

[18] Delivery reports must also be enabled for your company. If you are unable to send messages with delivery reports please contact your account manager to request delivery reports.

Within the M2MConnect platform these messages were originally not stored as distinct messages - the pertinent data was extracted from them and the details were updated in the database in the same record that held the original MT message.

Some customers are used to handling delivery reports as distinct messages, and requested that this functionality be implemented.

Accordingly, this has been implemented so that - for customers that desire this - Delivery Reports will be received as distinct messages.

There is no change to the way Delivery reports were previously treated - all customers are still able to view delivery reports via the present M2MConnect method, but additionally customers may receive Delivery Reports as MO messages.

These Delivery Reports are viewed or retrieved by customers (who have enabled this requirement) in the same way as they view / retrieve MO messages. Delivery Reports are differentiated from MO messages by means of the XML tag that wraps the message body, and by including the MT message reference within the message body:

e.g.

```
<messagerx>
        <sourcemsisdn>447973248421</sourcemsisdn>
        <destinationmsisdn>447817814107</destinationmsisdn>
        <receivedtime>14/12/2007 10:15:21</receivedtime>
        <bearer>SMS</bearer>
        <messageref>0</messageref>
        <message>Dev 1</message>
</messagerx>
```

and

```
<messagerx>
        <sourcemsisdn>447973248421</sourcemsisdn>
        <destinationmsisdn>447817814107</destinationmsisdn>
        <receivedtime>12/12/2007 14:34:24</receivedtime>
        <bearer>SMS</bearer>
        <messageref>0</messageref>
        <deliveryreport>SentMessageReference=28 id:1015920435 sub:001 dlvrd:001
        submit date:0712121639 done date:0712121639 stat:DELIVRD err:000
        text:reports as MO</deliveryreport>
</messagerx>
```

This setting too is found on the “maintain protocol” page, and ticking the check box will enable Delivery reports As MO (for those accounts that have Delivery reports enabled.)

MessageService WSDL (deprecated)
returnReports getDeliveryReports(username, password, deviceMSISDN)
MessageServiceByCountry WSDL
returnReports getDeliveryReports(username, password, deviceMSISDN,

```
                                        countryCode)

returnReports getDeliveryReportsFromDate(       username, password,
                                        deviceMSISDN,
                                        countryCode
                                        date, time )
```

String ***username***

M2MConnect username, used to validate access to the account. The user must have DATA access.

String ***password***

The M2MConnect password for the username. As all SOAP requests are transferred using HTTPS, this information is never sent across the network in plain text.

String ***deviceMSISDN***

The deviceMSISDN is an optional parameter and if specified (i.e. not empty) then only delivery reports from the specified MSISDN will be returned. This must be specified in international '+' format.

String ***countryCode***

When using the MessageServiceByCountry WSDL you can also filter based on the countryCode of the device. This allows you to retrieve delivery reports for each country separately. Country codes should be specified as the two or three digit prefix. For UK this is 44, for France 33, for Belgium 32.

String ***date***

When using the MessageServiceByCountry WSDL you can also filter based on the date of reception. This allows you to only retrieve the delivery reports received after the specified date. This field is in the format dd/mm/yyyy. If this field is left blank the report will include all records from the earliest date possible.

String ***time***

When using the MessageServiceByCountry WSDL with a date filter you can also add a filter based on the time of reception. This allows you to only retrieve the delivery reports received after the specified date and time. This time field is in the format dd/mm/yyyy hh:mm:ss, and must be specified in UTC. If the time component is not entered then the entire day will be included.

String [] ***returnReports***

This method will return an array of delivery reports in XML format. If no reports are available, then an empty array will be returned.

An M2MConnect delivery report will look like:

```
<deliveryreport>
        <messageref>64</messageref>
        <sourcemsisdn>441234567890</sourcemsisdn>
        <msgsent>31/12/2002 23:59:59</msgsent>
        <status>Received - Success</status>
        <receivedtime>31/12/2002 23:59:59</receivedtime>
</deliveryreport>
```

Each M2M delivery report will be wrapped in the following format:

Tag	Description

deliveryreport	The top level element of a delivery report.
messageref	message reference that has been assigned to the message for this msisdn
sourcemsisdn	The MSISDN of the device where the original message was sent. This MSISDN will have been validated by the EE network.
msgsent	Specifies the date and time tag when the message was sent to the device.
Status	The status of the sent message. Can contain either of the three possible values: Received – Failed, Received – Success, Not Received. If no status update is available from the device, then ‘Not Received’ is returned in this tag.
receivedtime	Specifies the date and time tag when the message was received by the device. Present only when the delivery report has been received.

5 DEVELOPING A DEVICE

5.1 WBXML

Wap Binary Extensible Mark-up Language (WBXML) is simply a specification for the encoding of XML documents into a compact binary representation. It is used by all WAP phones to ensure the efficient transmission of data.

It is an optional feature that you can enable on your M2MConnect account (under ▸ maintain protocol). It is most useful for compressing plain text when using SMS. The reduction in message size will allow you to put significantly more information in the same text message. Also, if you are using encoding, M2MConnect supports fragmented SMS messages.

It is recommended that your solutions use the KXML parser, however, any parser that conforms to the WBXML specification should be compatible.

http://wireless.java.sun.com/midp/ttips/compressxml/

5.1.1 Concept overview

The main premise of the specification is focused around the idea of tokens. Blocks of text within an XML document are seen as tokens, each of which can be assigned a unique octal value. The blocks include tag names, attribute names and attribute values. Tokens are stored as a strict set in a table and replaced throughout the encoded document by the octal values. Therefore, a token that occurs often will be entered into the table once and a single octal value will replace all occurrences of it within the encoded document. The token table can be defined external to the message within a WBXML Parser or sent internally as part of the encoded message.

External token tables are called Known Token Tables. These tables are effectively hard coded into the WBXML Parser and must be defined at both ends of the communication channel. They are pre-defined and hence place greater restrictions on the types of documents that can be encoded in the long term. The tokens within these tables are ‘known’ by the parser, which gives greater control over how they are encoded. The algorithms for assigning octal values can be changed to give the greatest reduction in document size hence making the transmission of documents over restricted communication channels, more efficient. The biggest advantage of Known Token Tables is the fact that information about the encoded tokens doesn’t get sent as part of the message, again greatly reducing the overall size of encoded documents.

Internal token tables are called String Tables. These are dynamically generated at the time of parsing an XML document, ready for encoding. If the parser doesn’t have access to known tables or the known tables don’t define a particular tag, the tag name is tokenised and placed in an array at the front of the encoded document (the string table). The offset within the array is used as the encoding value and all occurrences within the XML document are replaced by the offset value. This technique is extremely flexible because the XML documents can contain any tag and the parser will simply generate the encoding as required. String Tables do carry a heavy cost in terms of the size of encoded documents. Information about the tokens must be sent as part of the message and therefore the encoding is a lot less efficient. The parser at either end of the communication channel doesn’t need to know about tokens and their values but any applications supporting the messaging will need to know how to deal with the tags once the documents have been decoded.

5.1.2 Dictionary Management

The ▸**maintain protocol** option on the main web menu allows an administrative user to upload the WBXML compression dictionary to the Gateway.

Uploaded files will not be propagated to devices and therefore integrators must allow for manual changes to be made to devices when a new protocol is uploaded. Alternatively, devices can support dictionary subsets, as long as they don’t change the positional order.

☞ Dictionary maintenance is designed to be performed during a system outage, or as part of the initial design stage. Changing the contents of the dictionary, or enabling/disabling encoding requires a corresponding change to the devices. If you wish to use a combination of encoded and non-encoded messages, or you wish to have multiple dictionaries you should use multiple M2MConnect accounts.

Once you enable encoding **ALL** messages received for the account must be encoded using WBXML. Any messages that aren't WBXML encoded will be discarded, and an entry will be written to the system log.

If you disable encoding then all messages received will be passed through to the application. This may result in your application getting binary messages (ie, any WBXML encoded ones) that aren't anticipated.

5.1.3 Bearer Differences

You can use WBXML compression for both SMS and GPRS. All messages received on any bearer must be in the appropriate format based on the settings for the receiving account.

All WBXML messages sent using SMS need to use the SMS PDU mode. This allows the GSM module to directly send a string of binary data. The first three bytes of the SMS contain a M2MConnect header. This header includes the reply-to information as well as allowing for multiple part SMS messages.

	SMS	GPRS
WBXML	▪ PDU mode ▪ Up to 16 SMS 'fragments' per message ▪ proprietary M2MConnect fragmentation protocol ▪ Message references are supported	▪ up to 128 KB ▪ message references are available
XML	▪ up to 39015 characters per message ▪ 3gpp specified fragmentation protocol (www.3gpp.org) ▪ reply-to or message reference will always be 0	▪ up to 128 KB ▪ message references are available
Plain text	▪ up to 39015 characters per message ▪ 3gpp specified fragmentation protocol (www.3gpp.org) ▪ reply-to or message reference will always be 0	▪ up to 128 KB ▪ message references are available
Binary	▪ up to 140 bytes ▪ only one SMS per message ▪ reply-to or message reference will always be 0	▪ up to 128 KB ▪ message references are available

5.1.4 Sample compressed messages

Compressing the following message (340 characters) using the dictionary defined in 5.1.2 results in a message 92 bytes long.

```
<m2m>
  <servlet>
      <servlet-name>test 1</servlet-name>
    <servlet-class>test 2</servlet-class>
      <init-param>
         <param-name id="0.1">test 3</param-name>
         <param-value>test 4</param-value>
         <description name="Thing">test 5</description>
      </init-param>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <foo>
      <bar>test 6</bar>
      <gservlet-class>test 7</gservlet-class>
   </foo>
</m2m>
```

The hex values that would be transmitted using WBXML encoding, and the configured dictionary, is:

```
0101040050464503746573742031000147037465737420320001 48C9058501
03746573742033000 14A03746573742034000 1CB0686010374657374350001
014C03310001014D4E03746573742036000 14F03746573742037000 10101
```

If you were to use WBXML encoding, without using a dictionary you would transmit a message similar to the follow (270 bytes):

```
01010481046D326D00736572766C657400736572766C65742D6E616D65007
36572766C65742D636C61737300696E69742D706172616D00706172616D2D
6E616D6500696400706172616D2D76616C7565006465736372697074696F6
E006E616D65006C6F61642D6F6E2D737461727475700066 6F6F0062617200
6773657276 6C65742D636C617373004400440403200044 0C0374657374203
10001032000441903746573742032000103200044 27C432043D03302E3100
0103746573742033000144400374657374203400 01C44C0458035468696E6
700010374657374203500010320000103200044 5D0331000101446D032000
4471037465737420360001032000447503746573742037000 10101
```

As you can see the preconfigured dictionary can produce quite a substantial saving in space. In fact, simply using WBXML without a dictionary saves a lot of overhead.

5.1.5 M2MConnect dictionary files

Configuring the WBXML dictionary in M2MConnect is done using a text file. The default dictionary contains the following text:

```
tokens.tag.names=
tokens.attr.names=
tokens.attr.values=
```

The three lines represent placeholders for the known token table of tag names, attribute names and attribute values.

Using the XML example above, the configuration file would be as follows:

tokens.tag.names=**servlet-name,servlet,servlet-class,init-param,param-name,param-value,description,load-on-startup,foo,bar,gservlet-class,m2m**
tokens.attr.names=**id,name**

```
tokens.attr.values=0.1,Thing
```

The values (i.e. the comma delimited lists after the ‘=’ character) define the known token table used by the WBXML encoder. Known tokens are assigned in order, and because the known token table is position sensitive, must be in the same order on the device and the gateway.

If you wish to extend the known token table to support new tokens, you should add to the end of the list. This will ensure that existing devices can continue to encoding using their existing known token table [19].

There is only one known token file for each customer and all of the definitions for a customer’s devices will be placed within it. The string table definitions will need to be exactly the same at both ends of the communication channel. If they are different in any way the decoding of messages may not work correctly[20]. It is the responsibility of the user to ensure the devices are amended whenever a new protocol is uploaded.

The algorithm used to encode and decode messages from within KXML is an exact implementation of the WBXML specification (http://www.w3.org/TR/wbxml/) and therefore parsers that are developed for the devices also need to conform to the same specification.

The **attr.values** table applies to all the attributes in the XML. In addition, the attr.names table applies to all XML tags. This allows you to reuse an attribute name in many tags without increasing the size of the known token table.

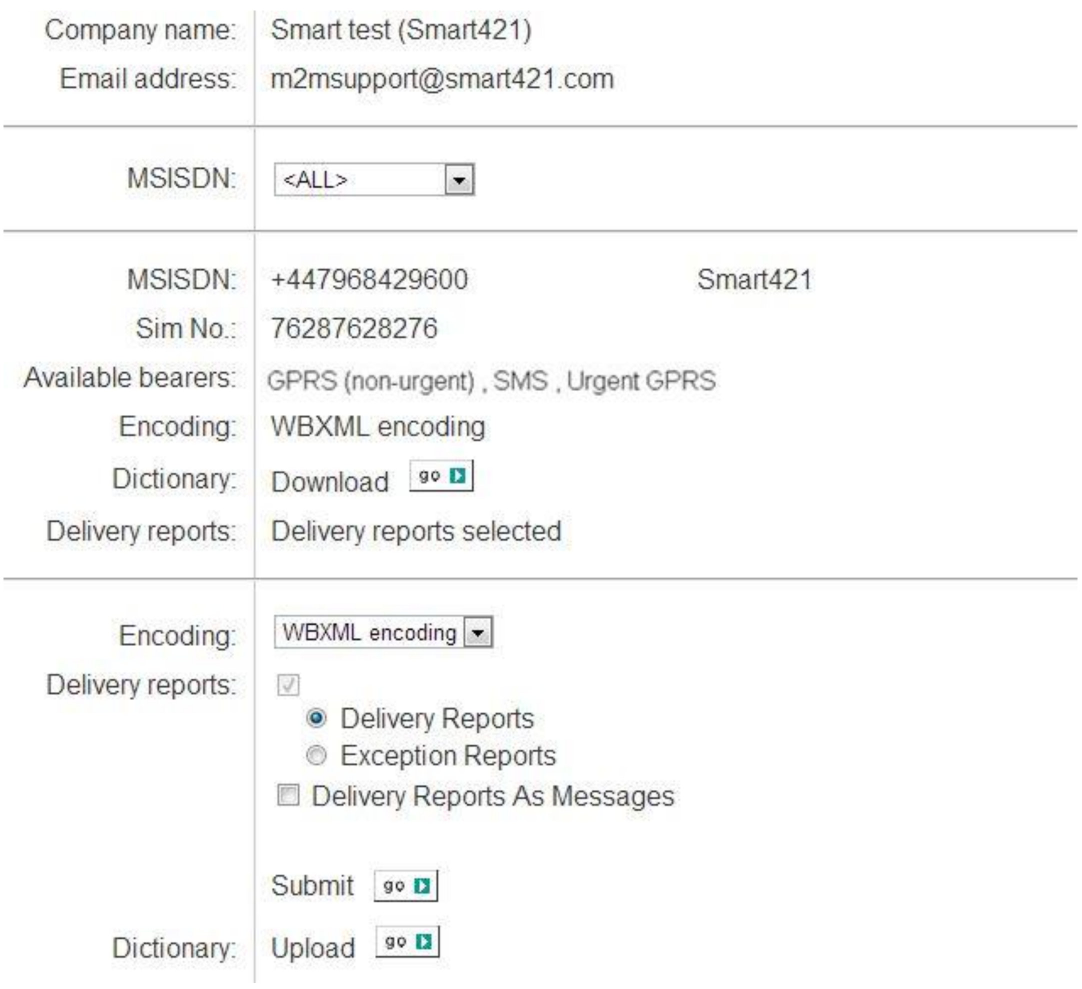
Any attribute names, values, or tags that are not defined in the known token table will be passed through into the WBXML encoded message as a free-form string table in the compressed message.

5.1.6 Uploading a dictionary

To upload a dictionary go into ▸**maintain protocol**. Under the field WBXML Encoding check to see if encoding is enabled.

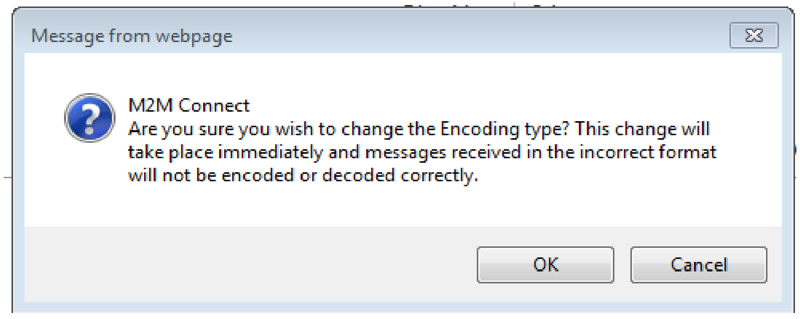
[19] However the application must ensure that it doesn’t send XML with the extended stringset to the device as it won’t have the new strings in it’s stringtable.

[20] Manifested by messages containing the wrong tags



Company name:	Smart test (Smart421)
Email address:	m2msupport@smart421.com
MSISDN:	<ALL>
MSISDN:	+447968429600 Smart421
Sim No.:	76287628276
Available bearers:	GPRS (non-urgent) , SMS , Urgent GPRS
Encoding:	WBXML encoding
Dictionary:	Download go
Delivery reports:	Delivery reports selected
Encoding:	WBXML encoding
Delivery reports:	✓ ● Delivery Reports ○ Exception Reports ☐ Delivery Reports As Messages
	Submit go
Dictionary:	Upload go

To configure a dictionary you must enable WBXML Encoding by selecting "WBXML Encoding" from the Encoding drop-down box. This will present a confirmation dialog box similar to the one below



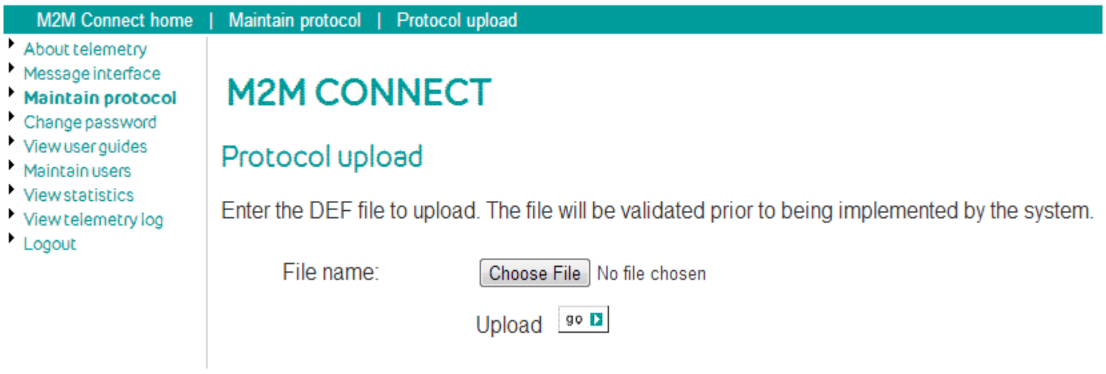
Message from webpage

M2M Connect
Are you sure you wish to change the Encoding type? This change will take place immediately and messages received in the incorrect format will not be encoded or decoded correctly.

Once you have confirmed this dialog box the maintain protocol screen will change to allow you to upload a dictionary:

Use Download go to download any pre-configured dictionary. For new accounts this will be an empty file that can be used as a template.

Once your dictionary is ready for uploading, click on the Upload go button. This will take you to the protocol upload screen:



Protocol upload

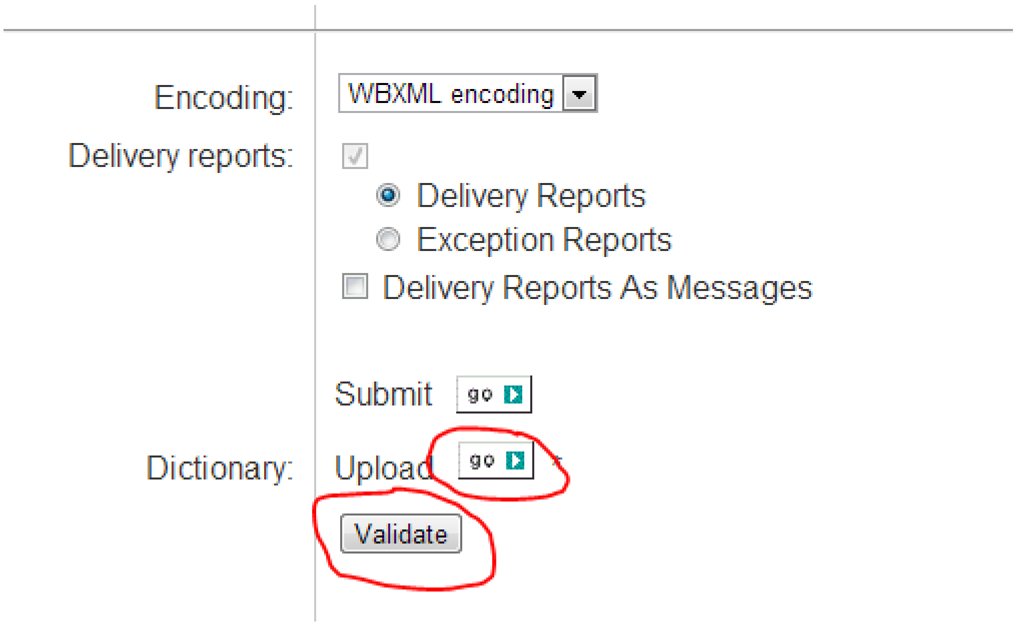
Enter the DEF file to upload. The file will be validated prior to being implemented by the system.

File name: No file chosen

Upload

Use Browse... to select the .def file, or type the file name into the text entry box, and then click on Upload go to send the configuration file to M2MConnect.

Once the configuration file has been successfully uploaded you need to validate it, which will confirm the contents and activate it on M2MConnect. This is done by clicking on the Validate button.



Encoding: WBXML encoding

Delivery reports: ✓

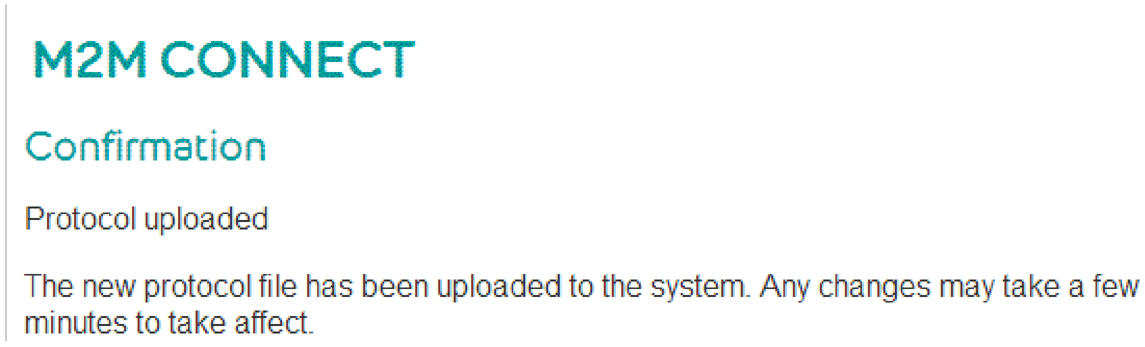
● Delivery Reports

○ Exception Reports

☐ Delivery Reports As Messages

Dictionary:

A confirmation screen will be displayed when the dictionary has been activated:



Confirmation

Protocol uploaded

The new protocol file has been uploaded to the system. Any changes may take a few minutes to take affect.

5.1.7 Compression tips

Tags to select a value, rather than indicate what the value is. For example:

```
<immobiliser>enabled</ immobiliser >
<immobiliser >disabled</immobiliser >

tokens.tag.names=immobiliser
tokens.attr.names=
tokens.attr.values=
```

Compresses less than:

```
<immobiliserEnabled/>
<immobiliserDisabled/>

tokens.tag.names=immobiliserEnabled,immobiliserDisabled
tokens.attr.names=
tokens.attr.values=
```

Or, if the known tag token table is becoming too large:

```
<immobiliser state="Enabled"/>
<immobiliser state="Disabled"/>


tokens.tag.names=immobiliser
tokens.attr.names=state
tokens.attr.values=Enabled,Disabled
```

5.1.8 Whitespace

M2MConnect will preserve all whitespace when sending message to your devices. To ensure the most efficient compression it is recommended that you strip all unnecessary whitespace before sending the message.

For example:

```
<servlet>¶

· <servlet-name>test 1</servlet-name>¶

</servlet>
```

Compresses to:

```
0101040046030D0A2000450374657374203100010030D0A0001
```

The 030D0A2000 and 030D0A00 are unnecessary characters.

If the whitespace had been removed before the message was compressed:

```
<servlet><servlet-name>test 1</servlet-name></servlet>
```

Compresses to the following, 20 byte shorter string:

```
0101040046450374657374203100001
```

5.1.9 Number formats & Binary Data

As is designed as a universal method of exchanging data it avoids many common integration problems caused by binary encoding of data. However, this can result in larger message formats….

Methods of reducing the overhead include:

- Base64 or Hex encoding of numbers
- Use of entities to directly code binary data into the XML string

5.1.10 Simplifications

When sending messages from the device it is possible to hard code the compressed string, and then just substitute the appropriate data. This is not a recommended way of processing received messages as the incoming message may contain string tables, and tags that aren't expected.

For example, with the following message, using the string table from the previous example:

```
<servlet-name>test 1</servlet-name>
```

This will compress to the following:

```
01010400450374657374203100 01
```

The 03 and 00 in the example above represent the start and end of an inline string[21]. All the text between these tags correspond to the first tag in the XML message

The string **746573742031** corresponds to the hex characters for the string 'test 1'. To send the same message replacing 'test 1' with 'livecode' you simply replace the contents between the 03 and 00:

```
010104004503 6C697665636f6465 0001
```

5.1.11 Trouble shooting

Several problems can manifest as a result of using dictionaries. The most likely is a dictionary mismatch. This happens when the dictionary configured on the M2MConnect gateway differs from the one used by the device. As a result, messages will be assigned the wrong tags & attribute values when they are decoded.

To prevent this you should ensure that you either use a separate M2MConnect account (and therefore dictionary) for major device versions, or only append to the known token tables.

[21] STR_I in the WBXML specification

6 SMS

Throughout the following sections:

- SMS refers to the short message that is sent over the GSM network.
- Message refers to the logical message sent through M2MConnect.
- All AT commands refer to the Sony Ericsson AT command set. This may be different for different devices.
- Sample protocol lines prefixed by '>' and in italics to the modem or device. Other lines are commands that the user would issue. In some cases the modem will issue a '>', which is then followed by user text. This will be indicated by an italic '>' followed by non italicised text.

6.1 First Steps

M2MConnect acts as a virtual mobile, allowing it to receive and send SMS messages. One of the first things you should try is to send an SMS from any EE phone to your gateway

The first step is to initialise the device to use the correct SMS mode for the type of messages you wish to send.

The command to select between different encoding modes is AT+CMGF

To select Plain Text SMS use **AT+CMGF=1**

```
AT+CMGF=1
>OK
```

To send WBXML encoded SMS you need to select PDU message mode. This is done using **AT+CMGF=0**

```
AT+CMGF=1
>OK
```

To determine the current messaging mode use **AT+CMGF=?**

```
AT+CMGF=?
>+CMGF: (0-1)
```

The above response indicates that both mode one and two is supported.

6.2 Plain Text SMS

6.2.1 Sending

To send a message using text mode XML, you need to:

1. Ensure that plain text mode has been selected. This only needs to be done once.
2. Use **AT+CMGC** to send the message.

```
AT+CMGF=1                          //Select plain text
>OK
AT+CMGC="07968429640"              //Send a message to the gateway
>Hello
[Ctrl+Z]
>+CMGS: 7                          //Response from the modem to indicate
>OK                                that it's been sent successfully
```

Once the modem has displayed the +CMGS response the message has been sent to the SMC. The actual number returned will increment.

6.2.2 Receiving Messages

To receive SMS messages, new message indication must be activated as follows:

```
AT+CNMI?
>+CNMI: 2, 0, 0, 0, 0
>OK
```

Once the message indication is activated, new messages receipt will be indicated by the following command:

```
>+CMTI: "SM",16
```

This command will be automatically generated, when the device receives any new message. The numeric value in this command, i.e. 16, indicates the memory location where the message is saved in the device. This message can be retrieved using the CMGR command:

```
AT+CMGR=16
>+CMGR: "REC UNREAD","447817814226",,"04/02/02,16:17:38+00"
>test message from server
>
>OK
```

6.3 PDU Mode SMS in WBXML

When using M2MConnect and WBXML encoded messages you need to use the GSM module’s PDU mode. This allows you to completely control the contents of the SMS message.

In addition to this all binary messages sent to M2MConnect have a 3 byte header prefixed to them. This header contains fields used by M2MConnect to provide message Ids, as well as supporting multiple fragments in a single message.

6.3.1 Receiving

A PDU string contains not only the message, but also a lot of meta-information about the sender, the SMS service centre number, the time stamp etc. It is in the form of hexadecimal *octets* and decimal *semi-octets*. The following is a PDU string returned from the modem for a WBXML encoded message:

```
0791449737019037040C914497862469040004403032512530211110000001010400450374657374203100001
```

The following table describes a break-up of all the octets contained in the PDU string.

Octets	Description
07	Length of the SMSC information. In this case it is 7 octets.
91	Type-of-address for service centre number. For international format, it is 91.
449737019037	service centre number.
04	First octet of this SMS-DELIVER message. The bit fields that make the value 04 are described below
0C	Address-Length.

`449786246904`	Source MSISDN Number. This is 44 79 68 42 96 40, it in the message stored as BCD, with the high and low digits reversed.
`00`	Protocol Identifier (PID)
`04`	Data Coding Scheme
`40 30 32 51 25 30 21`	Time stamp
`11`	Length of message, which is all data following from here (17 bytes = 3 for WBXML header + 14 for the message data)
`100000`	The WBXML message headers. This is a single fragment message, with a message id of 0.
`0101040045037465737420310001`	The message data – this is the message defined in section 5.1.10

Service Centre Number

The beginning contains the short message centre number. This number is stored as BCD, with the high and low digits reversed. Reversing the digits in 449737019037, you will get the service centre number. You should confirm that this is 447973100973 to ensure that nobody is spoofing numbers. This will ensure that you are receiving messages actually sent from the source phone number. The service centre number is represented in decimal semi-octets.

First octect of the SMS-DELIVER PDU message

This is the SMS-DELIVER information associated with the message. The first octet of the SMS-DELIVER PDU has the following layout.

SMS Deliver PDU								
Bit	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRI	Unused	Unused	TP-MMS	TP-MTI	TP-MTI

Field	Description
TP-RP	Reply path. Parameter indicating that reply path exists.
TP-UDHI	User data header indicator. For M2MConnect this is always 0
TP-SRI	Status report indication. This bit is set to 1 if a status report is going to be returned to the SME
TP-MMS	More messages to send. This bit is set to 0 if there are more messages to send. This is always 1 for M2MConnect.
TP-MTI	Message type indicator. Bits no 1 and 0 are both set to 0 to indicate that this PDU is an SMS-DELIVER

The first octet of the SMS-DELIVER PDU is specified as a hexadecimal value, in the PDU string. For example, a first octet of "04" (hexadecimal) has the following meaning:

SMS Deliver PDU								
Bit	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRI	Unused	Unused	TP-MMS	TP-MTI	TP-MTI
Value	0	0	0	0	0	1	0	0

Address Length

Length of senders MSISDN number specified in hexadecimal. As the length of senders number is 12 decimal, converted to hex it will be 0C.

Source MSISDN Number

This is the number of the application that sent the PDU SMS. This number is stored as BCD, with the high and low digits reversed. Reversing the digits in 449786246904, you will get the sender's number as 44 79 68 42 96 40. Source MSISDN number is represented in decimal semi-octets.

Protocol Identifier

The Protocol-Identifier parameter consists of one octet. The value 00 here represents a SMS Protocol based message. The mobile device should interpret reserved or unsupported values as the value 00000000 but shall store them exactly as received. The service centre may reject messages with a TP-Protocol-Identifier containing a reserved value or one which is not supported.

Data Coding Scheme

The Data-Coding-Scheme field, defined in GSM 03.40, indicates the data coding scheme of the actual message data. The hex value of 0x04 indicates the binary payload used for WBXML. Plain text messages would be 0x00.

Timestamp

The time stamp specifies the time when the message was sent by the application. This number is stored as BCD, with the high and low digits reversed. When reversed, equals "04 03 23 15 52 03 12", where the 6 first characters represent date, the following 6 represents time, and the last two represents time-zone related to GMT. The value comes out to be: 23-03-2004 15:52:03 GMT+3. The timestamp is represented in decimal semi-octets.

Length of Message

This is the length of PDU message. As the message content is 8-bit data, as specified in the Data-Coding-Scheme octet, the length of this message will be the number of octets. It includes the WBXML header, and the contents of the message.

WBXML Header

The payload section of the message, in which the actual data is transmitted, must have a specific format of numeric values in the first three octets. These values determine that the message is a part of fragment and its position in the chain of fragments. The payload header takes the following form:

First Octet								
Bit	7	6	5	4	3	2	1	0
Name	unused	unused	Response Indicator	Last Fragment Indicator	Fragment-Sequence-Number			

Second Octet								
Bit	7	6	5	4	3	2	1	0
Name	Message Reference Number (High Byte)							

Third Octet								
Bit	7	6	5	4	3	2	1	0
Name	Message Reference Number (Low Byte)							

The constituents of payload header are:

Section	Description

Unused Bits	Two bit numeric value. This is unused and must be specified as 00.
Response Indicator	Indicates whether the message is sent as a response to a request sent by the gateway, or is originating from the device. If the message is originating from the device, this bit should be 0, otherwise 1.
Last Fragment Indicator	Indicates whether the message contains more than one fragment. The last fragment indicator can be either 0 or 1. When a message is split into fragments, all but the last fragment is sent with the last fragment indicator set to 0. The last fragment is sent with the indicator set to 1.
Fragment Sequence Number	Indicates the position of the fragment in the chain. This can range from 0 to 15. M2MConnect supports a maximum of 16 fragments.
Message Reference Number	Provides a reference to the message that is fragmented. This can range from 1 to 65535.

All WBXML binary messages must include the payload header, even if it is a single fragment. Messages will not be processed by M2MConnect until all fragments have been received, including last fragment having Last Fragment Indicator set to 1. M2MConnect Gateway will log rejected messages where a complete fragment set has not been received within the Gateway Fragment-Timeout setting.

Individual messages fragments are referred to one single message using the Message Reference Number. This allows devices to implement a response to given message request, and helps in grouping of multiple SMS fragments to form a complete message over the 140 octet limit. M2MConnect will always assign a Msg-Reference[22] to the device, when sending a message to it. The number will be a cyclical value for each device MSISDN in the range 1 to 65535. If the number of messages sent to the device surpasses this range, within the archiving period (e.g. 10 days) then the additional messages will result in an error and be rejected by the M2MConnect.

While responding to a request, the message should contain Response Indicator having value set to 1. The Msg-Reference number should be the same as indicated in the received message. This functionality allows a client to send a message to a device and wait for a response. Using the SOAP interface this occurs automatically, provided the device returns the correct Msg-Reference and Response-Indicator.

If the message is being originated by the device, the Response Indicator should be set to 0 and Msg-Reference number should be a unique value. The Msg-Reference value cannot be re-used for a new message until the M2M Gateway has processed all fragments of the previous message reference.

Message Data

This is the WBXML encoded XML data.

6.3.2 Sending

To send WBXML compressed messages, the process is same as for Text mode SMS, however you must ensure that the encoding mode is set to 0:

```
AT+CMGF=0
>OK
```

[22] **Note**: Msg-Reference numbers sent by the Gateway, or a device, are not guaranteed to be without gaps in the sequence.

While sending the message, the length of the actual message string should be specified in octet. The following is an example PDU packet that would send a message to M2MConnect from a UK number.

```
0011000B914497862469040004AA1110000101010400450374657374420310001
```

Sending this message would be as follows:

```
AT+CMGF=0           // Set PDU mode (initial configuration)
>OK

AT+CSMS=0           // Check if modem supports SMS commands (initial configuration)
>OK

AT+CMGS=32          // Send message, 32 octets (excluding the two initial zeros)
>000011000B914497862469040004AA1110000101010400450374657374420310001
[Ctrl+Z]
>+CMGS: 7           // Response from the modem to indicate
                    // that it's been sent successfully (doesn't indicate delivery)
```

The following table describes a break-up of all the octets contained in the PDU string.

Octets	Description
00	Length of SMS Centre Number. This value is 0, which means SMSC information stored in the SIM will be used.
11	First Octet of the SMS-SUBMIT Message
00	Message-Reference. The "00" value here lets the mobile device set the message reference number itself.
0C	Address length of the destination number
91	Type-of-address for destination number. For international format, it is 91.
449786246904	Destination MSISDN Number. This is 44 79 68 42 96 40, it in the message stored as BCD, with the high and low digits reversed.
00	Protocol Identifier (PID)
04	Data Coding Scheme – WBXML binary PDU messages are 0x04
AA	Validity Period. Specifies the time when the message expires. AA means validity period is 4 days[23].
11	Length of message, including the WBXML header & the WBXML compressed data
100001	The WBXML message headers. This is a single fragment message, with a message id of 1.
0101040045037465737420310001	The message data – this is the message defined in section 5.1.10

All the octets in an outbound PDU SMS are same as in an inbound message, except the First Octet of the SMS-SUBMIT PDU Message.

SMS-SUBMIT PDU message

This is the SMS-SUBMIT information associated with the message. The first octet of the SMS-SUBMIT PDU has the following layout.

[23] Refer ETSI 03.40 for more information

SMS Submit PDU								
Bit	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRR	TP-VPF		TP-RD	TP-MTI	

Field	Description
TP-RP	Reply path. Parameter indicating that reply path exists.
TP-UDHI	User data header indicator. This bit is set to 1 if the message data field starts with a header
TP-SRR	Status report request. This bit is set to 1 if a status report is requested
TP-VPF	Validity Period Format. Possible combinations: 00 that means TP-VP field not present (use default), 10 implies TP-VP field present using relative format
TP-RD	Reject duplicates. Parameter indicating whether or not the service centre shall accept an SMS-SUBMIT PDU message having the same Message Reference as a previously submitted message from the same device.
TP-MTI	Message type indicator. Bits no 1 and 0 set to 0 and 1 respectively indicates that this PDU is an SMS-SUBMIT

The first octet of the SMS-SUBMIT PDU is specified as a hexadecimal value, in the PDU string. For example, a first octet of "11" (hexadecimal) has the following meaning:

SMS Submit PDU								
Bit	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRR	TP-VPF	TP-VPF	TP-RD	TP-MTI	
Value	0	0	0	1	0	0	0	1

Address Length

Length of destination MSISDN specified in hexadecimal. For UK numbers this length is 12 decimal, or 0x0C hex. This doesn’t include the ‘Type of Number’ field.

Type of Number (Destination)

This ETSI numbering scheme used for the destination number. The value for international format numbers is 0x91.

Destination MSISDN Number

This is the number of the application that will receive the PDU SMS. This number is stored as BCD, with the high and low digits reversed. Reversing the digits in 449786246904, you will get the sender’s number as 44 79 68 42 96 40. Destination MSISDN number is represented in decimal semi-octets.

Protocol Identifier

The Protocol-Identifier parameter consists of one octet. The value 00 here represents a SMS Protocol based message. The mobile device should interpret reserved or unsupported values as the value 00000000 but shall store them exactly as received. The service centre may reject messages with a TP-Protocol-Identifier containing a reserved value or one which is not supported.

Data Coding Scheme

The Data-Coding-Scheme field, defined in GSM 03.40, indicates the data coding scheme of the actual message data. The hex value of 0x04 indicates the binary payload used for Binary or WBXML. Plain text messages would be 0x00.

TP-VP (Validity Period)

The length of time this message is valid for. For example, using relative validity periods (selected by TP-VPF in the SMS-Deliver PDU byte), 0xAA indicates 4 days. Using validity periods the expiry time is calculated by the following:

TP-VP Value	Meaning	Example
0 to 143	(TP-VP + 1) * 5 minutes (i.e. 5 minutes intervals up to 12 hours)	0x0b = 60 minutes
144 to 167	12 hours + ((TP-VP - 143) * 30 minutes)	0x9b = 18 hours
168 to 196	(TP-VP - 166) * 1 day	0xa9 = 3 days
197 to 255	(TP-VP - 192) * 1 week	0xc3 = 3 weeks

Length of Message

This is the length of PDU message. As the message content is 8-bit data, as specified in the Data-Coding-Scheme octet, the length of this message will be the number of octets. It includes the WBXML header, and the contents of the message.

WBXML Header

See the section on WBXML header described under receiving messages.

Message Data

This is the WBXML encoded XML data.

6.3.3 SMS Message Fragmentation

M2MConnect supports different maximum size for messages depending on the encoding used by the messages. For example, unfragmented plain text XML messages can have a maximum size of 160 characters and unfragmented Binary encoded or WBXML messages can contain up to 140 bytes. However, sometimes the situation demands the use of larger messages that can carry more data than the maximum size. In order to overcome these limits, message fragmentation is available in M2MConnect.

6.3.3.1 WBXML

Using proprietary fragmentation, large messages can be split up into a series of messages that can be sent over the network and collected on the other end as a part of one single message. However, proprietary fragmentation is only available for WBXML messages.

M2MConnect defines a specific structure for fragmented messages; this is passed in the WBXML header.

- Fragments are connected together by the **message reference**. For fragmented messages you must use a unique message reference number. Numbers must remain unique at least until all fragments have been received.
- Each fragment should set which fragment number in the **fragment sequence number** field. The first fragment is numbered 0.
- The last fragment should have the **Last Fragment Indicator** set to true.

6.3.3.2 Plain Text

Plain text messages of up to 39015 characters can now be sent between M2MConnect and the device, depending upon the device capability. M2MConnect now implements the “Concatenated Short Messages” protocol defined in the SMPP v3.4 specification.

Note! This capability is device-dependant. The number of fragments in a Concatenated Short Message implementation varies amongst devices. Customers should ensure that overall message lengths conform to the capabilities of their devices, or data loss may occur.

6.4 PDU Mode SMS in Binary

When using M2MConnect and binary encoded messages you need to use the GSM module's PDU mode. This allows you to completely control the contents of the SMS message.

6.4.1 Receiving

A PDU string contains not only the message, but also a lot of meta-information about the sender, the SMS service centre number, the time stamp etc. It is in the form of hexadecimal *octets* and decimal *semi-octets*. The following is a PDU string returned from the modem for the binary message 0x00, 0x01, 0x02, 0xFF:

```
07914497370190370400C91449786246904000440303251253021040000102FF
```

The following table describes a break-up of all the octets contained in the PDU string.

Octets	Description
`07`	Length of the SMSC information. In this case it is 7 octets.
`91`	Type-of-address for service centre number. For international format, it is 91.
`449737019037`	service centre number.
`04`	First octet of this SMS-DELIVER message. The bit fields that make the value 04 are described below
`0C`	Address-Length.
`91`	Type-of-address of the Source MSISDN. For international format, it is 91.
`449786246904`	Source MSISDN Number. This is 44 79 68 42 96 40, it in the message stored as BCD, with the high and low digits reversed.
`00`	Protocol Identifier (PID)
`04`	Data Coding Scheme
`40 30 32 51 25 30 21`	Time stamp
`04`	Length of message, which is all data following from here (4 for the message data)
`000102FF`	The binary message data – 0x00, 0x01, 0x02, 0xFF

Service Centre Number

The beginning contains the short message centre number. This number is stored as BCD, with the high and low digits reversed. Reversing the digits in 449737019037, you will get the service centre number. You should confirm that this is 447973100973 to ensure that nobody is spoofing numbers. This will ensure that you are receiving messages actually sent from the source phone number. The service centre number is represented in decimal semi-octets.

First octect of the SMS-DELIVER PDU message

This is the SMS-DELIVER information associated with the message. The first octet of the SMS-DELIVER PDU has the following layout.

SMS Deliver PDU								
Bit	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRI	Unused	Unused	TP-MMS	TP-MTI	TP-MTI

Field	Description
TP-RP	Reply path. Parameter indicating that reply path exists.

TP-UDHI	User data header indicator. For M2MConnect this is always 0
TP-SRI	Status report indication. This bit is set to 1 if a status report is going to be returned to the SME
TP-MMS	More messages to send. This bit is set to 0 if there are more messages to send. This is always 1 for M2MConnect.
TP-MTI	Message type indicator. Bits no 1 and 0 are both set to 0 to indicate that this PDU is an SMS-DELIVER

The first octet of the SMS-DELIVER PDU is specified as a hexadecimal value, in the PDU string. For example, a first octet of "04" (hexadecimal) has the following meaning:

SMS Deliver PDU								
Bit	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRI	Unused	Unused	TP-MMS	TP-MTI	TP-MTI
Value	0	0	0	0	0	1	0	0

Address Length

Length of senders MSISDN number specified in hexadecimal. As the length of senders number is 12 decimal, converted to hex it will be 0C.

Source MSISDN Number

This is the number of the application that sent the PDU SMS. This number is stored as BCD, with the high and low digits reversed. Reversing the digits in 449786246904, you will get the sender’s number as 44 79 68 42 96 40. Source MSISDN number is represented in decimal semi-octets.

Protocol Identifier

The Protocol-Identifier parameter consists of one octet. The value 00 here represents a SMS Protocol based message. The mobile device should interpret reserved or unsupported values as the value 00000000 but shall store them exactly as received. The service centre may reject messages with a TP-Protocol-Identifier containing a reserved value or one which is not supported.

Data Coding Scheme

The Data-Coding-Scheme field, defined in GSM 03.40, indicates the data coding scheme of the actual message data. The hex value of 0x04 indicates the binary payload used for WBXML. Plain text messages would be 0x00.

Timestamp

The time stamp specifies the time when the message was sent by the application. This number is stored as BCD, with the high and low digits reversed. When reversed, equals "04 03 23 15 52 03 12", where the 6 first characters represent date, the following 6 represents time, and the last two represents time-zone related to GMT. The value comes out to be: 23-03-2004 15:52:03 GMT+3. The timestamp is represented in decimal semi-octets.

Length of Message

This is the length of PDU message. As the message content is 8-bit data, as specified in the Data-Coding-Scheme octet, the length of this message will be the number of octets.

Message Data

This is the data payload of the message.

6.4.2 Sending

To send binary messages, the process is same as for WBXML binary SMS, however you must ensure that the encoding mode is set to 0:

```
AT+CMGF=0
```

```
>OK
```

While sending the message, the length of the actual message string should be specified in octet. The following is an example PDU packet that would send a message to M2MConnect from a UK number.

```
0011000B914497862469040004AA04000102FF
```

Sending this message would be as follows:

```
AT+CMGF=0          // Set PDU mode (initial configuration)
>OK

AT+CSMS=0          // Check if modem supports SMS commands (initial configuration)
>OK

AT+CMGS=32         // Send message, 32 octets (excluding the two initial zeros)
>000011000B914497862469040000AA04000102FF
[Ctrl+Z]
>+CMGS: 7          // Response from the modem to indicate
                   // that it's been sent successfully (doesn't indicate delivery)
```

The following table describes a break-up of all the octets contained in the PDU string.

Octets	Description
00	Length of SMS Centre Number. This value is 0, which means SMSC information stored in the SIM will be used.
11	First Octet of the SMS-SUBMIT Message
00	Message-Reference. The "00" value here lets the mobile device set the message reference number itself.
0C	Address length of the destination number
91	Type-of-address for destination number. For international format, it is 91.
449786246904	Destination MSISDN Number. This is 44 79 68 42 96 40, it in the message stored as BCD, with the high and low digits reversed.
00	Protocol Identifier (PID)
04	Data Coding Scheme – binary PDU messages are 0x04
AA	Validity Period. Specifies the time when the message expires. AA means validity period is 4 days[24].
04	Length of message, which is all data following from here (4 for the message data)
000102FF	The binary message data – 0x00, 0x01, 0x02, 0xFF

All the octets in an outbound PDU SMS are same as in an inbound message, except the First Octet of the SMS-SUBMIT PDU Message.

SMS-SUBMIT PDU message

This is the SMS-SUBMIT information associated with the message. The first octet of the SMS-SUBMIT PDU has the following layout.

[24] Refer ETSI 03.40 for more information

SMS Submit PDU								
Bit	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRR	TP-VPF		TP-RD	TP-MTI	

Field	Description
TP-RP	Reply path. Parameter indicating that reply path exists.
TP-UDHI	User data header indicator. This bit is set to 1 if the message data field starts with a header
TP-SRR	Status report request. This bit is set to 1 if a status report is requested
TP-VPF	Validity Period Format. Possible combinations: 00 that means TP-VP field not present (use default), 10 implies TP-VP field present using relative format
TP-RD	Reject duplicates. Parameter indicating whether or not the service centre shall accept an SMS-SUBMIT PDU message having the same Message Reference as a previously submitted message from the same device.
TP-MTI	Message type indicator. Bits no 1 and 0 set to 0 and 1 respectively indicates that this PDU is an SMS-SUBMIT

The first octet of the SMS-SUBMIT PDU is specified as a hexadecimal value, in the PDU string. For example, a first octet of "11" (hexadecimal) has the following meaning:

SMS Submit PDU								
Bit	7	6	5	4	3	2	1	0
Name	TP-RP	TP-UDHI	TP-SRR	TP-VPF	TP-VPF	TP-RD	TP-MTI	
Value	0	0	0	1	0	0	0	1

Address Length

Length of destination MSISDN specified in hexadecimal. For UK numbers this length is 12 decimal, or 0x0C hex. This doesn’t include the ‘Type of Number’ field.

Type of Number (Destination)

This ETSI numbering scheme used for the destination number. The value for international format numbers is 0x91.

Destination MSISDN Number

This is the number of the application that will receive the PDU SMS. This number is stored as BCD, with the high and low digits reversed. Reversing the digits in 449786246904, you will get the sender’s number as 44 79 68 42 96 40. Destination MSISDN number is represented in decimal semi-octets.

Protocol Identifier

The Protocol-Identifier parameter consists of one octet. The value 00 here represents a SMS Protocol based message. The mobile device should interpret reserved or unsupported values as the value 00000000 but shall store them exactly as received. The service centre may reject messages with a TP-Protocol-Identifier containing a reserved value or one which is not supported.

Data Coding Scheme

The Data-Coding-Scheme field, defined in GSM 03.40, indicates the data coding scheme of the actual message data. The hex value of 0x04 indicates the binary payload used for Binary or WBXML. Plain text messages would be 0x00.

TP-VP (Validity Period)

The length of time this message is valid for. For example, using relative validity periods (selected by TP-VPF in the SMS-Deliver PDU byte), 0xAA indicates 4 days. Using validity periods the expiry time is calculated by the following:

TP-VP Value	Meaning	Example
0 to 143	(TP-VP + 1) * 5 minutes (i.e. 5 minutes intervals up to 12 hours)	0x0b = 60 minutes
144 to 167	12 hours + ((TP-VP - 143) * 30 minutes)	0x9b = 18 hours
168 to 196	(TP-VP - 166) * 1 day	0xa9 = 3 days
197 to 255	(TP-VP - 192) * 1 week	0xc3 = 3 weeks

Length of Message

This is the length of PDU message. As the message content is 8-bit data, as specified in the Data-Coding-Scheme octet, the length of this message will be the number of octets.

Message Data

This is the binary payload data.

6.5 Special

6.5.1 Alphanumeric MSISDN:

(Max 11 characters)

6.5.1.1 Description

Designate a Gateway account to a customer and substitute the GatewayMSISDN with an 11 character alpha-numeric string for use in OUK MT SMS.

6.5.1.2 Background

SMS MSISDNs can be represented by Alpha-numeric names, rather than numbers. These names are reserved and cannot be amended or shared amongst devices, so a specific MSISDN will always have the designated alpha-numeric name. This facility presents an opportunity for companies to send messages and have their name displayed as the source rather than an anonymous numeric. M2MConnect are able to set up accounts for specific customers so that MT messages sent from the M2MConnect platform are received by the customers' devices showing a name as the source.

6.5.1.3 Caveats

- This solution is limited to MT SMS messages through EE.
- These MT messages are one way only. A reply to the alpha-numeric MSISDN will NOT be received by M2MConnect.
- Similarly, MT messages with alpha-numeric source MSISDN will NOT receive Delivery Reports. These can be requested, but will not be received by M2MConnect.
- For an M2MConnect customer to use this facility will require specific tasks by EE. Please contact your EE Technical Consultant.

7 GPRS

GPRS (General Packet Radio Service) is a packet-based wireless communication service that does not require a continuous channel from a portable terminal for the transmission and reception of data. It makes very efficient use of available radio spectrum, delivering "always-on" wireless packet data services EE M2MConnect offers data rates from 9.6 up to 43 Kbps.

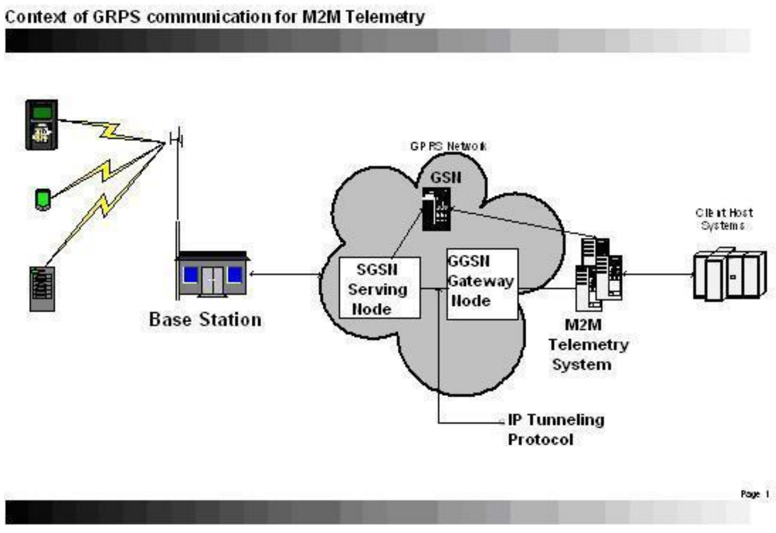
EE M2MConnect provides a GPRS gateway for routing telemetry messages over GPRS. With the use of M2MConnect, remote devices are addressed transparently by an MSIDSN.

7.1 GPRS Fundamentals

7.1.1 GGSN & SGSN

GSN (GPRS Support Nodes) are the elements needed to implement GPRS functionality in an existing GSM network. These support nodes comprises two nodes: Serving GPRS Support Node (SGSN) and Gateway GPRS Support Node (GGSN). The SGSN is responsible for tasks like mobility management, session management, compression of data, user authentication, encryption, tunnelling and routing of data. The GGSN is responsible for enabling the connection of a remote GPRS device to M2MConnect.

The following figure shows the M2MConnect GPRS Architecture:



Broadly speaking, the SGSN deals with the radio interface and GGSN deals with the server end. When roaming for example, you would use your local GGSN and the SGSN of the network you are roaming on.

7.1.2 Device MSISDN

For the BEEP communications you need to know the MSISDN number assigned to the mobile device. To do this you use the AT+CNUM command[25].

```
AT+CNUM
>+CNUM: "Data","+447970842295", 145          The MSISDN along with other output parameters.
>OK
```

[25] If this command is not successful, it is always possible to send a short message to another phone or establish a call in order to retrieve the MSISDN of your device.

The command returns three parameters. The first parameter is an optional alphanumeric string associated with the MSISDN. The second parameter is the MSISDN of the mobile device. The third parameter is the type of address octet in integer format[26].

7.1.3 APN

M2MConnect uses an APN (Access Point Name) to route GPRS traffic to the M2MConnect gateway. This APN ensures that M2MConnect traffic is separate from other GPRS traffic, and that devices are not Internet accessible.

To connect to the M2MConnect APN you need to have a M2MConnect GPRS Talkplan. These can be activated on existing M2MConnect SIM. The next step is to configure the GSM Module so that it knows how to talk to the APN.

```
AT+CGDCONT=1,"IP","m2m.orange.co.uk"Add the M2MConnect APN
>OK
AT*ENAD?                          Get a list of APNs configured.
>*ENAD: 1,"GPRS 1","",,1,0,0,1,0  This is the M2M APN. The first number is called the CID
                                  (context identification parameter) and is used to connect
                                  to GPRS. You use it below to specify the username and
                                  password for the APN.
>OK
AT*ENAD=1,"GPRS 1","m2mconnect","m2mconnect",1,0,0,1,0
>OK                               The bold 1's in the command are the CID listed
                                  above.
```

Once the APN has been configured onto the module you should be able to attach to it, configuring a PDP context.

7.1.4 PDP Context

When a device connects to a GPRS APN it establishes a PDP Context.

This context is stored in the mobile device, the SGSN, and the GGSN. With an active PDP context, the mobile device is able to send and receive data packets. You can query the PDP context stored in the mobile device using the AT+CGDCONT command as follows:

```
AT+CGDCONT?
>+CGDCONT: 1, "IP", "m2m.orange.co.uk", "", 0, 0    Here the first parameter, i.e. 1 is
                                                    the context id required to activate
                                                    the PDP Context
>OK
```

You can then activate the PDP context using the AT+CGACT command. This command accepts two parameters: Context ID (CID), and PDP State. The PDP State parameter indicates the state of PDP Context. The following example shows how to activate a PDP context.

```
AT+CGATT=1
>OK

AT+CGDCONT?
>+CGDCONT: 1, "IP", "m2m.orange.co.uk", "", 0, 0
>OK
```

[26] For more information about this parameter, refer GSM 04.08[13] sub clause 10.5.4.7

AT+CGACT=1,1	Activate the PDP context having CID as 1, the second parameter specifies this is a command to activate the PDP Context. For deactivating this parameter will be 0.
>OK	

7.1.5 IP Addressing

EE M2MConnect uses private dynamic IP addressing for devices. Dynamic addressing is based on the idea of IP reuse. An address is assigned when a device attaches to the network, it is retrieved when the devices disconnects. This address is then assigned to some other device connecting to the network.

Dynamic IP addressing is used to efficiently utilize a limited number of available IP addresses on the network provider’s end.

In order to start transmission over GPRS, the device must make a BEEP connection with EE M2MConnect on a specific IP/Port address. The connection should be made at 10.168.50.75 on port 10288.

Hereafter, if the device needs to send any message to the application, it uses BEEP to send message to the MSIDSN number of the application.

7.1.6 Security in GPRS

The security features in M2MConnect GPRS network are similar to those for GSM. They include:

Identity Confidentiality

It provides privacy to the customer by making it difficult to intercept the communication between the device and the GPRS network. For this purpose, standard GSM security methods are used.

Identity Authentication

Identity authentication is specified in the GSM standards. Authentication is performed in the SGSN, where pairs of random numbers and signed responses are stored. SGSN then uses an authentication to verify this value.

Data Confidentiality

It provides security and confidentiality of the user data transmitted between the remote device and the network. The information is protected by the use of encryption to make it unreadable by eavesdroppers.

Device Verification

All M2MConnect messages are associated with a verified device MSISDN. This verified MSISDN is used by M2MConnect to ensure that devices are only able to retrieve their messages. It is also used to put the sender into the received message.

MSISDN cross-check

As an additional check, the BEEP protocol requires the device to identify itself. This is then confirmed with the MSISDN that the GPRS network has reported. Device programmers can use this to ensure that SIM cards have not been changed in the device[27].

[27] Possibly by reading the device MSISDN at initial configuration

7.2 BEEP Overview

GPRS is a data bearer that provides a channel for data transfer. To be able to develop applications over GPRS, one needs to define a protocol. This protocol is built on top of TCP/IP and is responsible for defining how the device talks to server.

BEEP (Block Extensible Exchange Protocol) defines a generic protocol framework for connection-oriented, asynchronous interactions. The basic concept of this protocol is to provide a simple and easy to implement protocol that can be easily used with low power clients. BEEP uses TCP as the underlying transport service, and is based on RFC 3080 of the Internet Engineering Task Force (IETF).

7.2.1 BEEP Architecture

The BEEP protocol is implemented on top of TCP/IP. Similar to application protocols such as FTP or TELNET, BEEP initializes a session for data exchange. Once a session is initialized, BEEP is responsible for maintaining the data communication, framing and multiplexing multiple channels in a single socket connection.

Session

A session forms the core of BEEP communication, and corresponds to a TCP/IP socket connection. A BEEP session allows asynchronous communication between remote devices and M2M Gateway. BEEP sessions contain logical channels

Channel

A channel is a logical conversation between the remote device and the M2MConnect Gateway. There are multiple channels in a BEEP session, all transported over a single TCP/IP connection.

M2MConnect normally uses the following channels:

Channel Number	Name	Usage
0	Control	Opening and Closing Channel 1 and 2, also used for the initial protocol negotiation
1	Transmit	Opened by the device to send messages to the server (MO). Is also used to request that M2MConnect send waiting messages to the device
2	Receive	Opened by M2MConnect in response to the device requesting delivery of MT messages

Frame

A BEEP frame is a block of information sent between the device and M2MConnect. Frames correspond with a physical communication over the TCP/IP socket.

7.2.2 Frames

Each frame contains which BEEP channel it is for, as well as information on how many bytes are being transmitted. Frames are organised into three segments; a header, the payload, and a trailer.

These segments store information about the data. The header and trailer segments contain data that that carry control information for that particular frame. The payload segment carries the actual data in form of octet-stream. Each of these segments in the frame is separated by a Carriage Return Line Feed (CRLF) combination[28].

Frame Header

[28] Or in HEX: 0x0D 0x0A

The Frame Header is responsible for describing the contents of the message and associating them with the appropriate logical channel. For example:

```
MSG 0 1 . 52 120
```

This information is transferred as text, with ASCII spaces (0x20) between the components.

Contents	Name	Description
`MSG`	Message Type	The first three bytes of the BEEP frame indicate what type of frame it is. This example is MSG type, meaning data being sent. Valid Message Types are: MSG, RPY, ERR, SEQ
`0`	Channel Number	The logical channel that this message relates to. Channel 0 represents the control channel. Odd numbers indicate channels opened by the device and even numbers indicate channels opened by the M2MConnect Gateway.
`1`	Message Number	Uniquely identifies a message and is used by the RPY and ANS functions. Message numbers can range from 0 to 2147483647. This number should sequentially increase for each message on the channel
`.`	Continuation Indicator	Indicates whether the message is split over more than one frame. The continuation indicator can be either an asterisk (*), or a period (.). When a message is split into multiple frames, all but the last frame is sent with the continuation indicator set to (*). The last frame is sent with the indicator set to (.). This is similar to message fragmentation in SMS.
`52`	Sequence Number	The sequence number is a non negative 32 bit integer that specifies the number of transmitted bytes that preceded the first byte in the payload for the specified channel. A number of 52 indicates that before this frame 52 bytes of payload data were transmitted on this channel. Sequence numbers are specific to the sender, so the device needs to count the total number of bytes it has transmitted on each channel. Additionally, a device should confirm that the received sequence number matches the total number of bytes already transmitted by M2MConnect to ensure that no packets have been dropped.
`120`	Size	The size of the payload in bytes. This does not include the BEEP header or trailer, or the CRLF that follows the BEEP header.

Payload

The BEEP frame payload is the largest section of a BEEP frame. Depending on the Message Type and channel this will contain different things. The exact contents of the Payload are described in the message scenarios listed below. Payloads do not need to have a trailing CRLF as the exact length is specified as the Size in the Frame Header.

Trailer

A BEEP frame trailer signifies the completion of a frame. It consists of the keyword END terminated with a CRLF character. The END keyword must be specified at end of every BEEP frame.

The BEEP specification defined in RFC 3080 recommends that in the absence of a trailer, the session can be terminated and a diagnostic entry logged. This is because a frame not containing a trailer may not have been correctly received and should be considered as being malformed by the receiving device or application.

7.2.3 *SEQ* Frames

Every BEEP frame sent on a BEEP channel has a sequence number. The sequence number of the first payload octet of the first data frame is 0 and initial window[29] size is 4096 octets. Once the channel is created, a device can update the window size by sending a SEQ frame. A SEQ frame is different from data frame, as it does not carry any payload data. Instead, it is used to synchronise flow of messages, and ensure that neither the client or the server is overloaded by a large transmission.

The format of a SEQ frame is as follows:

```
SEQ channel ackno window CR LF
```

A SEQ frame contains three elements, these are:

Element	Description
channel	A numeric value indicating the channel number.
Ackno	Indicates the value of the next sequence number that the sender is expecting to receive on this channel.
window	Indicates the window size, i.e. the number of payload octets, that the sender is expecting to receive on this channel. This is the largest frame size that can be transmitted to the sender on this channel.

The minimum frame size supported by both the device and M2MConnect must be 4096 bytes. This ensures that any message up to 4KB can be transmitted in one ‘chunk’.

Before sending a message, the device must ensure that the size of the payload is within the size advertised by the M2MConnect Gateway. Initially, this size is 4096. If the payload size is greater than this value, the device can segment the messages into smaller frames. Alternatively, the device can reduce the payload size by truncating any redundant octets that are accompanying the message.

If M2MConnect needs to send a message to the device that would exceed the advertised frame size it will be split into multiple BEEP frames.

Finally, BEEP frames must not cross the frame window boundary. When the sequence number on the channel plus the message size would cross the window boundary the transmission must be split into two separate BEEP frames.

7.3 Communicating with BEEP

The first step in a BEEP transaction is to create a connection with the GPRS network, that is, M2MConnect Gateway. First specify the PDP context and activate it.

The following example shows how to initiate a GPRS call from a device. It assumes that the APN has already been configured on the device (as CID 1):

[29] Similar to TCP/IP, BEEP follows a window-based flow control. Each channel has a sliding window that indicates the number of payload octets that a peer may transmit before receiving further permission.

```
AT+CGATT?        Check if the modem is attached to GPRS
>+CGATT:0        Return CGATT:0 implies that the modem is not attached
>OK

AT+CGATT=1       Log on to the network
>+CGATT:1        Successfully logged into the network
>OK

ATD*99***1#      Initiate a GPRS session to the APN with CID 1
```

Once the GPRS connection is established, the device should create a BEEP TCP/IP socket with M2MConnect on the following IP address and Port:

```
IP Address: 10.168.50.75
Port: 10288
```

Once a TCP/IP BEEP socket connection with M2MConnect is established, the device then creates a session and starts sending and receiving messages.

7.3.1 M2MConnect Fields

In order to allow for correct routing of messages and to allow only the correct application to collect messages intended for it, all messages must contain sufficient information to identify the intended recipient and optionally, the message being responded to.

In the following protocol flow scenarios you will see these fields used:

orig-msisdn

This element contains the MSISDN of the device[30] or application that is sending the message. When the device is sending to M2MConnect this is the device MSISDN. For messages received by the device this is the M2MConnect Gateway MSISDN.

dest-msisdn

This element contains the MSISDN of the device or application that receives the message.

resptomsg

This value specifies whether this is a response to a message or the message is originated for the first time. 0 implies that device is originating an unsolicited message, otherwise this contains the message reference

7.3.2 Creating a Session

When a BEEP session is established, the M2MConnect Gateway signifies its availability by immediately sending a positive reply with a message that contains a "greeting" element. The following example shows how a BEEP session is established between the device and the M2M Gateway.

```
  | Server to Client (131 bytes) This is the server announcing itself
S | RPY 0 0 . 0 109
S | Content-Type: application/beep+xml
S |
S | <greeting><profile uri='com.orange.m2m.smartbeep.profile' /></greeting>END
  | Client to Server (111 bytes) This is the client telling the server that it
  | supports the M2MConnect DeviceProfile service.
C | RPY 0 0 . 0 90
C | Content-Type: application/beep+xml
C |
C | <greeting><profile uri='DeviceProfile' /></greeting>END
C | Client to Server (16 bytes) This is the client announcing its transmission
C | window. This communication is optional, as long as the client never transmits
```

[30] To determine the MSISDN number of the device, see section 7.1.2

C	more than 4096 bytes on the control channel. It can be sent in the same socket
C	write as the previous contents as this saves one data packet being transmitted
C	(and the server will always announce a 109 byte frame)
C	SEQ 0 109 4096
	Server to Client (15 bytes) This is the server announcing its transmission
	window. Note that the sequence number is the same as the size from the
	previous BEEP Frame
S	SEQ 0 109 4096

In the above listing, `S` (server) signifies M2MConnect and `C` (client) denotes the device. Note that all communication so far has been on channel 0, the control channel. The Sequence Number for the first message in both directions is 0, as this is the first communication on channel 0 by either the server or the device.

One other thing of note is that either the client or the server can transmit first. The order of the communications above is not important, with the exception on the server sent SEQ frame. This SEQ will always follow the client write. Theoretically the client should not send an SEQ until it has received data from the server; however M2MConnect will always send a 109 byte greeting.

7.3.3 Creating Channels

For the device intends to create a channel, it must send a message containing the “start” element at channel 0.The start element contains a number attribute, that specifies the channel number[31] to be used. This is shown in the following listing:

	This is a MSG Frame on the control channel. It will request a data channel for
	sending messages to M2MConnect.
C	MSG 0 1 . 90 113
C	Content-Type: application/beep+xml
C	
C	<start number='1'><profile uri='com.orange.m2m.smartbeep.profile'/></start>END
	The response from the server may include a Framing SEQ. It will definitely
	include a reply that confirms that the M2MConnect profile is in use for the
	new channel. This indicates that the channel has been created correctly.
S	SEQ 0 203 4096
S	RPY 0 1 . 109 88
S	Content-Type: application/beep+xml
S	
S	<profile uri='com.orange.m2m.smartbeep.profile' />END

7.3.4 Receiving Messages

The device may request to start a download of any inbound messages waiting for collection at the Gateway, by sending a SBMSG_REQ message to the Server. The server will respond with a BEEP RPY message and will open a new channel using the M2MConnect profile, already published by the device. Once the channel is successfully opened, the server will then start sending the queued messages to the device. These messages will be of type SBMSG. The device must acknowledge each message by sending a RPY BEEP message to the system.

	110 Bytes

[31] **Note**: To avoid conflict in assigning channel numbers when requesting the creation of a channel, a device uses only positive integers that are odd-numbered. Similarly, M2MConnect uses only positive integers that are even-numbered. Therefore, as a rule of thumb, you should specify an odd number while requesting a channel creation from the device.

```
  | This is the request from the device (client) to download messages from
  | M2MConnect. It specifies the receving device MSISDN as the orig (it is sending
  | the request), and the M2MConnect GW MSISDN as the dest-msisdn
C | MSG 1 1 . 0 79
C |
C | msg-class=SBMSG_REQ
C | orig-msisdn=+447811160014
C | dest-msisdn=+447817814211
C |
C | END
  | 143 bytes
  | The server confirms the request, and then sends a channel open request to the
  | device. This may be transmitted as one or two TCP/IP packets, in this example
  | it has been transmitted as one. It is still two BEEP Frames, the confirmation
  | on channel 1, and a request to open channel 2 that is sent on channel 0
S | RPY 1 1 . 0 4
S |
S | OKEND
S | MSG 0 1 . 197 96
S | Content-Type: application/beep+xml
S |
S | <start number='2'><profile uri='DeviceProfile' /></start>END
  | As in the previous section, the client confirms the opening of the transmit
  | channel.
C | RPY 0 1 . 203 69
C | Content-Type: application/beep+xml
C |
C | <profile uri='DeviceProfile' />END
```

At this point M2MConnect will start to send messages waiting messages to the telemetry device. Each message sent follows the following exchange, simply increasing the message numbers and sequence numbers as appropriate for the flow. If no messages are waiting then the server will jump immediately to sending the NMTS message described below.

```
  | This is M2MConnect transmitting the first waiting message to the device. It is
  | sent on channel 2. Not that the orig-msisdn is the Gateway MSISDN and the
  | dest-msisdn is the device MSISDN
S | MSG 2 1 . 0 107
S |
S | orig-msisdn=447817814211
S | dest-msisdn=447811160014
S | resptomsg=1
S | msg-class=SBMSG
S |
  | <setTemp>20</setTemp>END
  | The device needs to confirm the receipt of this message to ensure that the
  | appropriate delivery report is generated. This will also stop M2MConnect from
C | retrying the send
C | RPY 2 1 . 0 4
C |
  | OKEND
```

When the system has finished sending all the messages an NMTS message will be sent to the device. The device should acknowledge this message. Once a reply to this message has been received by the Gateway from the device, the data channel will be closed.

```
  | The server tells the client (device) that there are no more messages waiting
  | for it.
S | MSG 2 2 . 107 20
S |
S | msg-class=NMTS
S |
S | END
  | Finally the client confirms that it has received the NMTS message allowing the
  | server to close the channel.
C | 300: Client to Server (24 bytes)
C | RPY 2 2 . 4 4
C |
C | OKEND
  | Once this has been received the server will close the transmit channel (2)
S | MSG 0 2 . 293 69
```

```
S | Content-Type: application/beep+xml
S |
S | <close number='2' code='200' />END
  | And the client should confirm it
C | RPY 0 2 . 272 44
C | Content-Type: application/beep+xml
C |
C | <ok />END
```

7.3.5 Transmitting a message

Once the transmit channel has been opened by the device, it can start transmitting message to the M2M Gateway. For sending outbound messages, the device must send a SBMSG message to the Gateway. On a successful receipt, the Gateway will respond with a RPY message containing an "OK"[32] element. The following example shows how to send a message from a device, using channel number 1:

```
  | The client sends the message.
  |        orig-msisdn contains the devices phone number.
  |        dest-msisdn is the M2MConnect gateway MSISDN
  |        resptomsg is either 0, or the message reference for a response
  |        msg-class is SBMSG to indicate a data message being sent
  | These are followed by an empty line to indicate the end of the MIME headers,
  | and then the contents of the M2MConnect message
C | MSG 1 2 . 79 107
C |
C | orig-msisdn=+447811160014
C | dest-msisdn=+447817814211
C | resptomsg=0
C | msg-class=SBMSG
C |
C | <overHeatingFault/>END
  | M2MConnect will acknowledge the message by responding with an OK
S | RPY 1 2 . 4 4
S |
S | OKEND
```

Once the message transmission is complete and there are no more messages to be sent, the device should send a NMTS message to the gateway as shown in the following example:

```
  | This stops the transfer session. The server should respond with an OK, though
  | it may simply terminate the session
C | MSG 1 3 . 186 20
C |
C | msg-class=NMTS
C |
C | END
```

7.3.6 Closing a Channel

Finally, the device should close channel 1 by sending "<close number='1'>" BEEP Frame to channel 0. This is formatted in the same way as M2MConnect closes a send channel, described above.

[32] **Note:** Every SBMSG message is acknowledged with a RPY message containing an "ok" element. This signifies the receipt of message on the Gateway. However, if there is no response affirming the receipt, it means that the message is not delivered and hence a duplicate message must be sent. Additionally, it is recommended that a diagnostic entry be logged.

7.4 Asynchrony

M2MConnect supports asynchronous BEEP interactions, both within a single channel and between separate channels.

Synchronous Communications

The device always initiates the transfer of messages. This means that you can sequentially send all outstanding messages, and then request any waiting messages to be downloaded[33]. This will ensure that there is never more than one message exchange active at once.

Asynchrony within a Single Channel

In order to improve performance a device can send multiple messages over a single channel without waiting for the corresponding replies. These can also be packaged as a single TCP/IP send reducing the number of transmitted packets.

The Gateway processes all the messages in the same order as they are received from a given channel and will generate replies as the messages are received and in the same order.

Asynchrony between Different Channels

A device can communicate on multiple channels at the same time without waiting for corresponding replies. This process is useful while sending SBMSG and SBMSG_REQ messages.

For example, a device can request the download of messages using SBMSG_REQ and then immediately start uploading messages on channel 1.

7.5 Message Segmenting

M2MConnect BEEP messages are enclosed in a message frame. Typically, small messages are carried in a single frame. However, it is also possible to split a single message into multiple frames. It is recommended that large messages be segmented into multiple frames to avoid transmission overheads. A segmented message is distinguished by a continuation character in the message header. A frame header containing a segmented message will look like:

```
MSG 2 3 * 433 364
```

The “*” in the header is a Continuation Indicator and it indicates that the given frame is a part of a message segmented into multiple frames. The last frame of the segment chain should have the Continuation Operator as “.”. This implies that the given frame is the last frame of the segment chain.

The following example shows how a message is split up into multiple frames:

```
  This stops the transfer session. The server should respond with an OK, though
  it may simply terminate the session
C MSG 1 1 * 0 4096
C data
C END
C MSG 1 1 . 4096 10
C Data
C END
```

It is important to note that message data must not cross the frame window boundary. Therefore when the sequence number plus the message size would exceed the boundary it should be split into two BEEP frames. In this situation a SEQ message **must** be sent between the first and second BEEP frame or the transfer won’t be accepted.

33 Prior to sending the NMTS message

7.6 Urgent GPRS

In situations when the application needs to send urgent data to a device but the device is not connected to M2MConnect, an application can send using “Urgent GPRS”.

This will cause M2MConnect to queue the message for normal GPRS transmission and also send an SMS message containing the plain text “GPRS COMMUNICATION REQUIRED”, all in uppercase. On receiving this message, the device should make a GPRS connection to M2MConnect.

7.7 Error Handling in BEEP

Error processing plays an important role in application development. However, error processing can be problematical in the wireless environment, as a loss of communication will mean that error conditions may go unnoticed. Some undocumented errors may occur given the nature of WAN communication. Therefore, to handle errors arising due to communication failure between the Gateway and device, the device end error processing should be based on the following principals.

- Whenever an error occurs, an ERR message should be sent from one party to the other. It follows that each listener will also need to intercept error messages as they arrive.
- Only messages that have been positively acknowledged should be deemed as sent. This means that unacknowledged messages may be resent. Each application needs to be aware of this condition and take whatever action is necessary to prevent duplicate processing.
- For communication errors, a timeout value of at least 120 seconds should be adopted after which, the connection should be considered dropped and all channels, connections, listeners etc should be terminated. A new connection should be initiated to make a “fresh start”.

8 GETTING HELP

In order to help solve problems quickly please ensure you have the following information ready before you call for support:

- Device MSISDN
- **M2MConnect** Gateway MSISDN
- Exact description of error, including any error codes returned or displayed
- Time the error was first encountered, and most recently occurred
- Which direction communication is failing (MO or MT)
- Which bearer is in use (SMS or GRPS)
- The last time the device succeeded in communicating
- Postcode for the device, or the last known location if a postcode isn't known

Additionally, if a third party, such as a device integrator, needs to contact EE for assistance with your SIM cards please ensure that a Third Party Consent form has been completed[34]. EE is unable to discuss accounts with people who are not authorised users.

[34] Please contact your account manager if you need further details