Introduction

1.1 Background

Short Message Service (SMS) is a feature in GSM telecommunications that allows short, textual messages to be delivered to cellular telephones.

SMS messages are transmitted over the control network, Signaling System 7 (SS7), and not the bandwidth channels allotted to voice communications. SS7 is the basis for all control networks used by all major GSM and wire line telephone carriers.

SMS has several advantages. It is more discreet than a phone conversation, making it the ideal form for communicating. SMS is a store-and-forward service, In addition to person-to-person messages, SMS can be used to send a message to a large number of people at a time, either from a list of contacts or to all the users within a particular area. This service is called broadcasting and is used by companies to contact groups of employees or by online services to distribute news and other information to subscribers.

Giving the PC ability to manage SMS, i.e. (send, receive, etc...) Will open a whole new opportunities of creating new services, commercial, bulk SMS, alarm systems … etc.

This feature is doable since many mobile and satellite transceiver units support sending and receiving of SMS using an extended version of the Hayes command set, a specific command language originally developed for the Hayes Smartmodem 300-baud modem in 1977.

1.2 Problem Statement

Managing (SMS) by using AT commands Technology using different type of software and languages

1.3 Objective

Building a Table of comparison, for the different applications that used the AT commands.

1.4 Methodology

We will send AT Commands from Application at the TE to the GSM modem (MS), and read the Command result, we will be using the AT commands that are related to SMS, which here will our area of interest.

The design is divided into two elements, hardware and software.

The hardware components:

1- USB GSM Modem (Huawei 3G).

2- PC (TE)

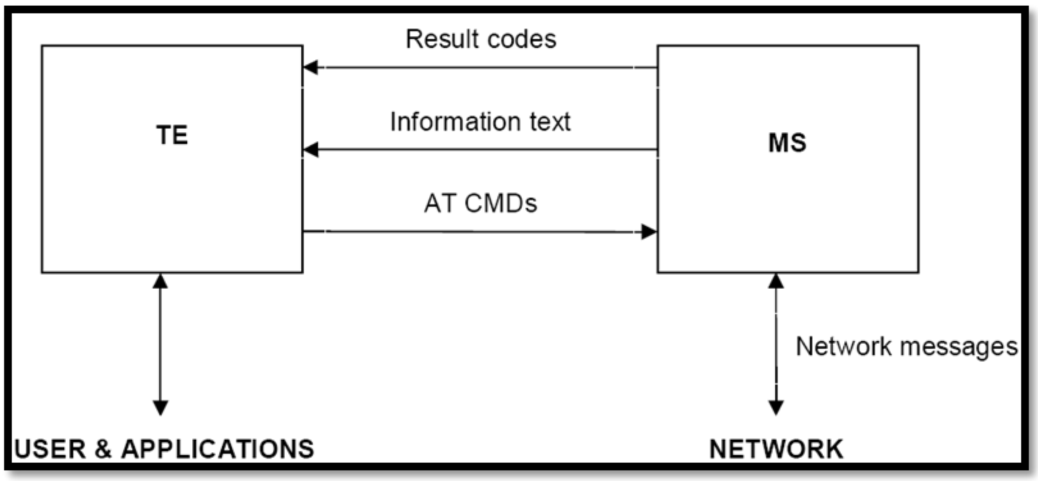


FIGURE 1.1: Block diagram of interaction between TE and MS

The software:

MATLAB R2009a

Visual Studio 2012 C#

Visual Studio 2012 VB

1.5 Research plan

The details of the chapter, as follows:-

Chapter two:

Literature and review of GSM Technology, short messaging service and the AT Command technology.

Chapter three:

Case studies

Chapter four:

Results and discussion

Chapter five:

Conclusion and Recommendations.

References.

2.Literature review

2.1 GSM

GSM (Global System for Mobile Communications: originally from Groupe Spécial Mobile) is the most popular standard for mobile telephony systems in the world. The GSM Association, its promoting industry trade organization of mobile phone carriers and manufacturers, estimates that 80% of the global mobile market uses the standard.[1] GSM is used by over 1.5 billion people across more than 212 countries and territories its ubiquity enables international roaming arrangements between mobile network operators, providing subscribers the use of their phones in many parts of the world. GSM differs from its predecessor technologies in that both signaling and speech channels are digital, and thus GSM is considered a second generation (2G) mobile phone system. This also facilitates the wide-spread implementation of data communication applications into the system

Figure 1.5: The GSM logo is used to identify compatible handsets and equipment

GSM network is a cellular network, which is a radio network distributed over land areas called cells, each served by at least one fixed-location transceiver known as a cell site or base station. When joined together these cells provide radio coverage over a wide geographic area. This enables a large number of portable

[1]

transceivers (e.g., mobile phones, pagers, etc.) to communicate with each other and with fixed transceivers and telephones anywhere in the network, via base stations, even if some of the transceivers are moving through more than one cell during transmission .

2.1.1 Evolution of mobile networks

I. First-Generation Mobile Systems

The first generation of analog cellular systems included the Advanced Mobile Telephone System (AMPS)1 which was made available in 1983. A total of 40MHz of spectrum was allocated from the 800MHz band by the Federal Communications Commission (FCC) for AMPS. It was first deployed in Chicago, with a service area of 2100 square miles2. AMPS offered 832 channels, with a data rate of 10 kbps. Although omnidirectional antennas were used in the earlier AMPS implementation, it was realized that using directional antennas would yield better cell reuse. In fact, the smallest reuse factor that would fulfill the 18db signal-to-interference ratio (SIR) using 120-degree directional antennas was found to be 7. Hence, a 7-cell reuse pattern was adopted for AMPS. Transmissions from the base stations to mobiles occur over the forward channel using frequencies between 869-894 MHz. The reverse channel is used for transmissions from mobiles to base station, using frequencies between 824-849 MHz.

In Europe, TACS (Total Access Communications System) was introduced with 1000 channels and a data rate of 8 kbps. AMPS and TACS use the frequency modulation (FM) technique for radio transmission. Traffic is multiplexed onto an FDMA (frequency division multiple access) system. In Scandinavian countries, the Nordic Mobile Telephone is used.

II. Second-Generation Mobile Systems

Compared to first-generation systems, second-generation (2G) systems use digital multiple access technology, such as TDMA (time division multiple access) and CDMA (code division multiple access). Global System for Mobile Communications, or GSM3, uses TDMA technology to support multiple users.

Examples of second-generation systems are GSM, Cordless Telephone (CT2), Personal Access Communications Systems (PACS), and Digital European Cordless Telephone (DECT4). A new design was introduced into the mobile switching center of second-generation systems. In particular, the use of base station controllers (BSCs) lightens the load placed on the MSC (mobile switching center) found in first-generation systems. This design allows the interface between the MSC and BSC to be standardized. Hence, considerable attention was devoted to interoperability and standardization in second-generation systems so that carrier could employ different manufacturers for the MSC and BSCs.

In addition to enhancements in MSC design, the mobile-assisted handoff mechanism was introduced. By sensing signals received from adjacent base stations, a mobile unit can trigger a handoff by performing explicit signaling with the network.

Second generation protocols use digital encoding and include GSM, D-AMPS (TDMA) and CDMA (IS-95). 2G networks are in current use around the world. The protocols behind 2G networks support voice and some limited data communications, such as Fax and short messaging service (SMS), and most 2G protocols offer different levels of encryption, and security. While first-generation systems support primarily voice traffic, second-generation systems support voice, paging, data, and fax services.

III. "2.5G" Mobile Systems

The move into the 2.5G world will begin with General Packet Radio Service (GPRS). GPRS is a radio technology for GSM networks that adds packet-switching protocols, shorter setup time for ISP connections, and the possibility to charge by the amount of data sent, rather than connection time. Packet switching is a technique whereby the information (voice or data) to be sent is broken up into packets, of at most a few Kbytes each, which are then routed by the network between different destinations based on addressing data within each packet. Use of network resources is optimized as the resources are needed only during the handling of each packet.

The next generation of data heading towards third generation and personal multimedia environments builds on GPRS and is known as Enhanced Data rate for GSM Evolution (EDGE). EDGE will also be a significant contributor in 2.5G. It will allow GSM operators to use existing GSM radio bands to offer wireless multimedia IP-based services and applications at theoretical maximum speeds of 384 kbps with a bit-rate of 48 kbps per timeslot and up to 69.2 kbps per timeslot in good radio conditions. EDGE will let operators function without a 3G license and compete with 3G networks offering similar data services. Implementing EDGE will be relatively painless and will require relatively small changes to network hardware and software as it uses the same TDMA (Time Division Multiple Access) frame structure, logic channel and 200 kHz carrier bandwidth as today's GSM networks. As EDGE progresses to coexistence with 3G WCDMA, data rates of up to ATM-like speeds of 2 Mbps could be available.

GPRS will support flexible data transmission rates as well as continuous connection to the network. GPRS is the most significant step towards 3G.

IV. Third-Generation Mobile Systems

Third-generation mobile systems are faced with several challenging technical issues, such as the provision of seamless services across both wired and wireless networks and universal mobility. In Europe, there are three evolving networks under investigation: (a) UMTS (Universal Mobile Telecommunications Systems), (b) MBS (Mobile Broadband Systems), and (c) WLAN (Wireless Local Area Networks).

V. Fourth generation of cellular wireless

It is a successor to 3G and 2G families of standards. which refers to all-IP packet-switched networks, mobile ultra-broadband (gigabit speed) access and multi-carrier transmission.[citation needed] Pre-4G technologies such as mobile WiMAX and first-release 3G Long term evolution (LTE) have been available on the market since 2006 and 2009 respectively.

4G also refer to IMT-Advanced (International Mobile Telecommunications Advanced), as defined by ITU-R. . An IMT-Advanced cellular system must have target peak data rates of up to approximately 100 Mbit/s for high mobility such as mobile access and up to approximately 1 Gbit/s for low mobility such as nomadic/local wireless access, according to the ITU requirements. Scalable bandwidths up to at least 40 MHz should be provided.

In all suggestions for 4G, the CDMA spread spectrum radio technology used in 3G systems and IS-95 is abandoned and replaced by frequency-domain equalization schemes, for example multi-carrier transmission such as OFDMA. This is combined with MIMO (i.e., multiple antennas(Multiple In Multiple Out)), dynamic channel allocation and channel-dependent scheduling.

2.1.2 The GSM network

The GSM network architecture (see Figure2) comprises several base transceiver stations (BTS), which are clustered and connected to a base station controller (BSC). Several BSCs are then connected to an MSC. The MSC has access to several databases, including the visiting location register (VLR), home location register (HLR), and equipment identity register (EIR). It is responsible for establishing, managing, and clearing connections, as well as routing calls to the proper radio cell. It supports call rerouting at times of mobility. A gateway MSC provides an interface to the public telephone network.

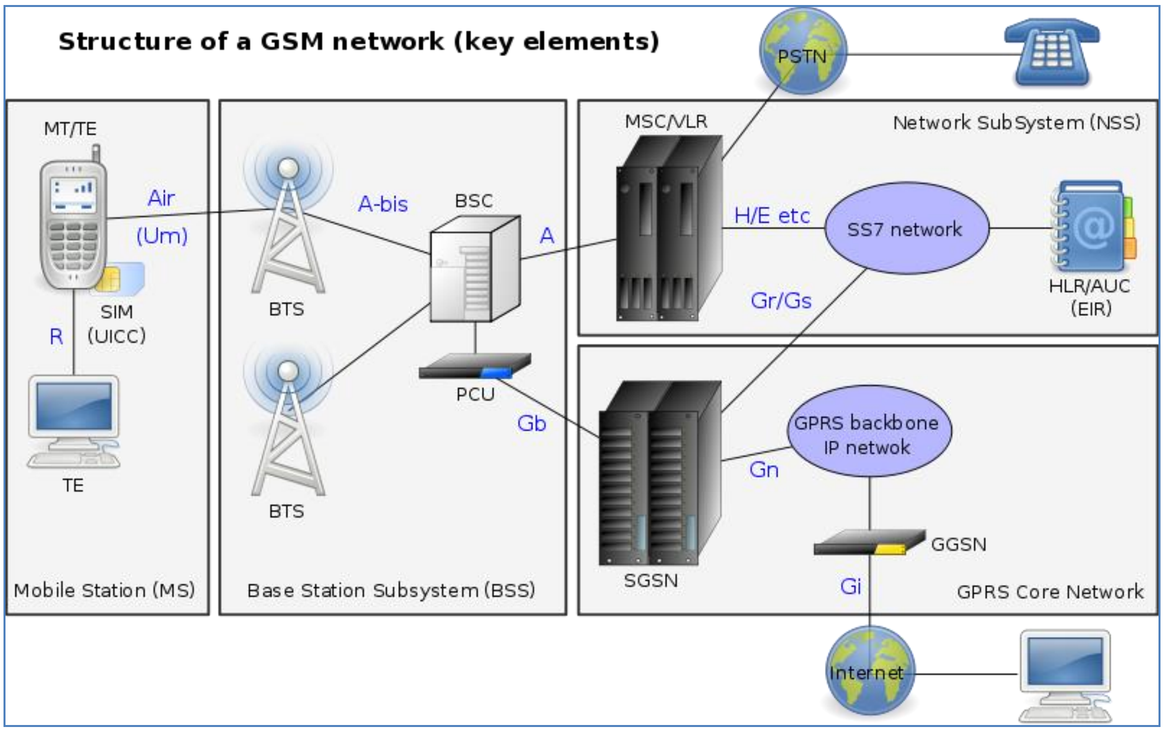


Figure 2.1: The GSM network architecture

The GSM divides the infrastructure into the following three parts.

• Network Switching Subsystems (NSS)

• Base Station Subsystem (BSS)

• Network Management Subsystem (NMS)

If we count the Mobile Station (MS) or cell-phone is the 4th element. Any telecommunications network requires some kind of NMS. A part of NMS is generic for any telecom system. The billing and messaging are two examples. The core of the NSS is the MSC (Mobile Switching Center) which is basically a PSTN switch with mobility management related enhancement/add-on. The BSS is entirely new (compared to PSTN) that are required for wireless access and mobility.

I. The Switching subsystem (NSS)

Fundamentally, the network and switching subsystems (NSS) is responsible for call connection, supervision and release operations between calling and called stations, where one or both of them are mobile stations (MS). Other functions include:

• Handling short messages and packet data (email, fax and a variety of notifications)

• Providing ‘bearer’ channel for data communications

• Maintaining database of its own users as well as visitors

• Variety of authentication and encryption

• Gateway to PSTN, other mobile networks and data networks including the Internet

Home Location Register

The home location register (HLR) is a database used for storing and managing subscriptions. Generally a PLMN (Public Land Mobile Network) consists of several HLRs. The first two digits of the mobile directory number (e.g. 0171 2620757) are the number of the HLR where the mobile subscriber is stored. The data includes permanent data on subscribers (such as subscriber's service profile) as well as dynamic data (such as current location and activity status). When an individual buys a subscription from one of the GSM operators, he or she is registered in the HLR of that operator.

Mobile Switching Center (MSC) and Visitor Location Register (VLR)

The mobile switching center (MSC) performs the telephony switching function. A mobile station must be attached to a single MSC at a time (either homed or visitor), if it is currently active (not switched off). The visitor location register (VLR) is a database attached to an MSC to contain information about its currently associated mobile stations (not just for visitors).

Note: A basic switch (that is a PSTN/ISDN switch) already has a database for its telephone connections. However, it is not designed to include visitors since a visitor has telephone number that does not belong to this switch. That is why a separate VLR is needed. An MSC, with the help of the HLR, allocates a visitor a ‘local’ telephone number (the MSRN), which is not currently allocated to anyone. This allocation is temporary (like visitor ID card). The VLR stores the MSRN as mobile station’s telephone number (along with other information). However, VLR also stores some information like ‘security triple’(authentication and encryption information) for each mobile station that are currently attached to the MSC. A VLR stores such information not only for its visitors but also for the homed mobile stations. From this perspective VLR is for homed mobile stations as well.

Authentication Center (AUC)

The authentication center (AUC) provides authentication and encryption parameters that verify the user's identity and ensure the confidentiality of each call. The AUC protects network operators from different types of fraud found in today's cellular world. The GSM has standard encryption and authentication algorithm which are used to dynamically compute challenge keys and encryptions keys for a call.

Equipment Identity Register (EIR)

The equipment identity register (EIR) is a database that contains information about the identity of mobile equipment that prevents calls from stolen, unauthorized, or defective mobile stations. The AUC and EIR can be implemented as stand-alone nodes or as a combined AUC/EIR node.

Base Station Subsystem (BSS)
All radio-related functions between mobile stations and network are performed in the base station subsystem (BSS).

The BSS consists of:

Base Transceiver Station (BTS)
A Base Station Transceiver (BTS) is a radio transceivers station that communicates with the mobile stations. Its backend is connected to the BSC. More detail about BTS will be covered later. A BTS is usually placed at the center of a cell. Its transmitting power defines the size of a cell. There are more on this later.

Base Station Controller (BSC)
A Base Station Controller (BSC) is a high-capacity switch with radio communication and mobility control capabilities. The functions of a BSC include radio channel allocation, location update, handover, timing advance, power control and paging.

Transcoder/Rate Adaptation Unit (TRAU)
The Transcoder/Rate Adaptation Unit (TRAU) is the data rate conversion unit. The PSTN/ISDN switch is a switch for 64 kbps voice. Current technology permits to decrease the bit-rate (in GSM radio interface it is 13 kbps for full rate and 6.5 kbps for half rate). Since MSC is basically a PSTN/ISDN switch its bit-rate is still 64 kbps. That is why a rate conversion is required in between the BSC and MSC (see the figure below)

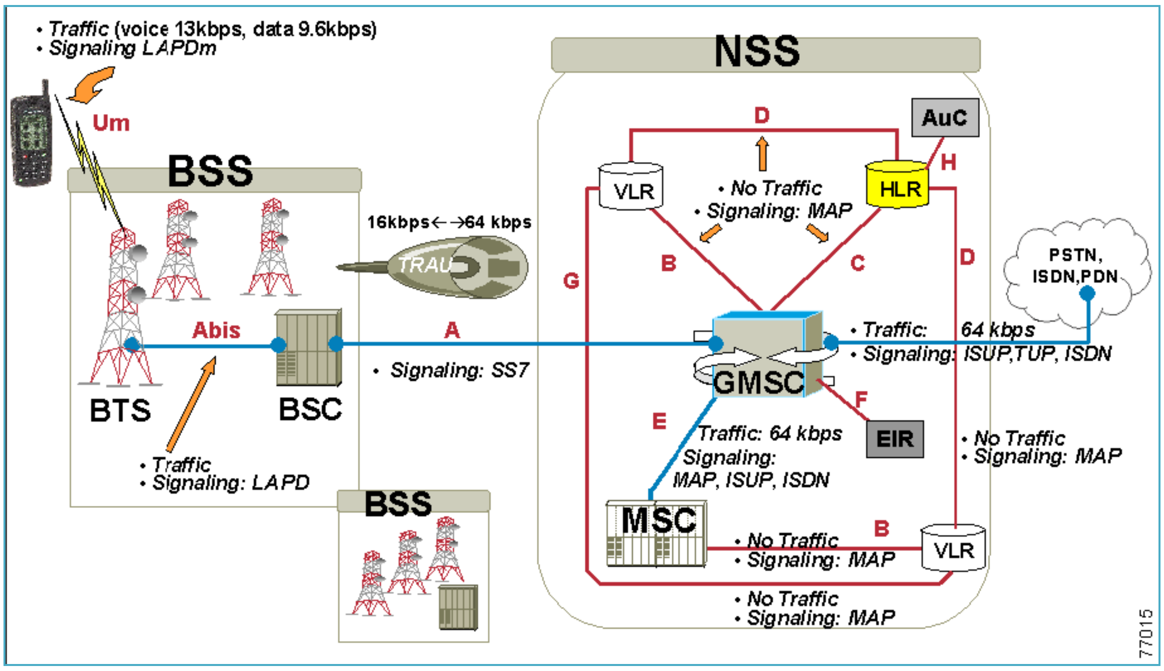


Figure 2.2: The Trans-coder/Rate Adaptation Unit

II. Network Management Subsystem (NMS)

Network Management Subsystem (NMS) includes network management center (NMC), operations and maintenance center (OMC) and a variety of other functions (see the network infrastructure diagram at the beginning of this document). A telecommunications network requires some kind of NMS. A part of NMS is generic for any telecom system.

Operation and Support System (OSS)

The operation support system (OSS) is to do a variety of operation and maintenance works such as commissioning and integrating new network elements to the existing system, software upgrade, collecting network performance statistics, reconfiguring network dimension and frequency planning.

Short Message Center (SMC)

The SMC, also MXE, is a node that provides integrated voice, fax, and data messaging. Specifically, the MXE handles short message service, cell broadcast, voice mail, fax mail, e-mail, and notification. And it will be discussed in more elaboration in the coming sections.

Intelligent Network

Like PSTN/ISDN the GSM system supports AIN (Advanced Intelligent Network) services through its mobile intelligent services node (MISN). This enables an operator to develop an innovative service and deploy that in its network. The principle was that of separation of the logic controlling services from the 'basic' call control function in existing telephone exchanges.

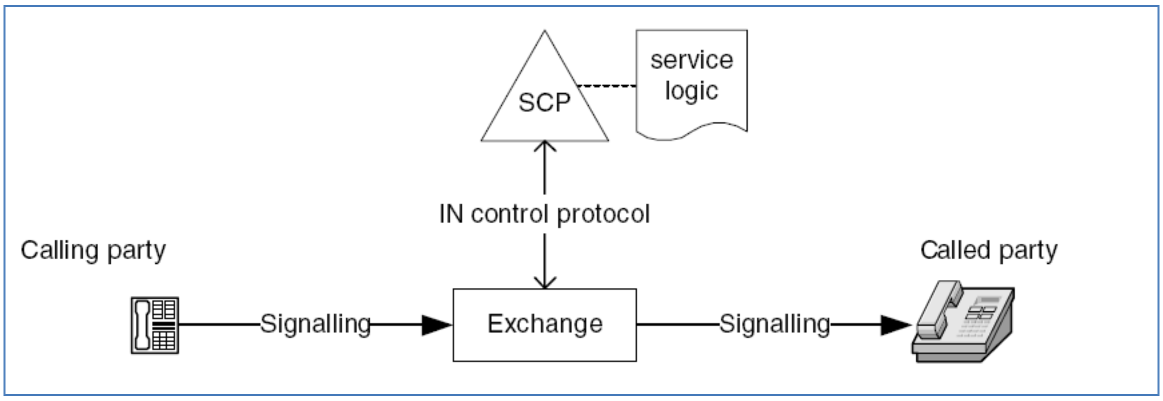


Figure 2.3: IN control basic call

The SCP forms the control platform for IN. The IN control protocol is the capability set that enables the operator to assert control over the call. Various IN standards have defined an IN protocol.

2.2 SMS Service

Short Message Service (SMS) is a relatively new feature in wireless telecommunications that allows short, textual messages to be delivered to cellular telephones. SMS is extremely popular in Europe and is gaining popularity in the US and worldwide. SMS messages are transmitted over the control network, Signaling System 7 (SS7), and not the bandwidth channels allotted to voice communications. SS7 is the basis for all control networks used by all major wireless and wire line telephone carriers. Disruption of SS7 operations could be devastating to the PSN and to NS/EP. SMS and related services, these additional features will greatly increase the resource requirements over SMS and, when combined with increased popularity, will impact the future load on SS7.

2.2.1 SMS Network element

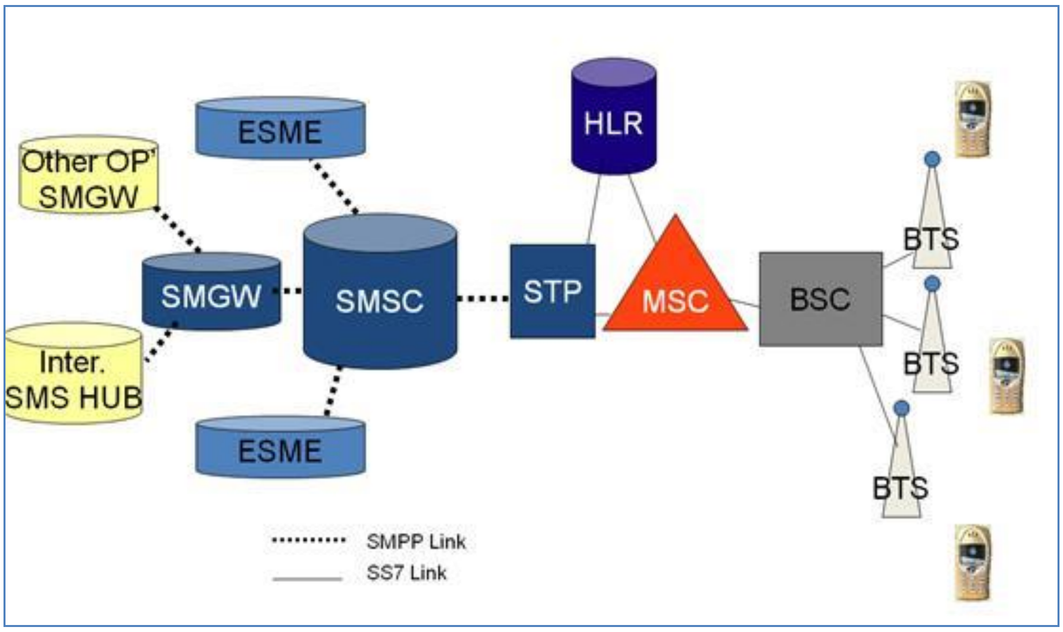


Figure 2.5: SMS Network elements

Typical SMS network consist of:

BSS

All radio-related functions between mobile stations and network are performed in the base station subsystem (BSS). Consists of:

(TRAU), (BTS) and (BSC), discussed in detail in section (1.1.3 The GSM network) subsection II.

NSS

The network and switching subsystems (NSS) is responsible for call connection, supervision and release operations between calling and called stations, where one or both of them are mobile stations (MS), also for delivering between the MS, SMC and other Network element such HLR, BSC and MSC.

SMC

The Short Messages Center, also known as Short Messages Service Center (SMSC). It is the network node that store and forwards the subscribers SM. It also connect the SMEs, SMGW with PLMN. The main role for the SMC is to store the messages from the sender, charge it and then route the message to the correct destination at the right time according to internal routing tables and scheduling mechanism.

An SMS center (SMSC) is responsible for handling the SMS operations of a wireless network:

When an SMS message is sent from a mobile phone, it will reach an SMS center first.

The SMS center then forwards the SMS message towards the destination.

The main duty of an SMSC is to route SMS messages and regulate the process. If the recipient is unavailable (for example, when the mobile phone is switched off), the SMSC will store the SMS message. It will forward the SMS message when the recipient is available.

Validity Period of an SMS Message - An SMS message is stored temporarily in the SMS center if the recipient mobile phone is offline. It is possible to specify a cutoff period after which the SMS message will be deleted from the SMS center. Once deleted, the SMS message will no longer be available for dispatch to the recipient mobile phone (even if it becomes online).

The SMC uses the SMPP protocol for communicating with any node, one of the STP functions in the SMS network is to act as translation point between MAP/ss7 to IP based messages which it is used in the typical SMC's.

SMGW

Is a node that connect an existing SMC of one operator with other operators SMC's. SMGW it is newly introduced and currently it is the dominant of the SMS interconnection traffic since it is based on IP connections it is so cheap in compare to the regular SS7 traffic through voice signaling gateways. SMGW is the answer to the question that always arise whenever there is disconnection in the PSTN network that separate one network SG from establishing calls with one or all operator. Still you can still find SMS traffic. SMGW also communicate through SMPP protocol. And has ability buffer and forward, not like the SMC which can store for long period and forward according to specific predefined plan.

ESME

External Short Message Entity (ESME), any node that able to communicate through SMPP, by mean of initiating an application layer connection with an SMSC over a TCP/IP network connection and then send and receive short messages to and from the SMSC respectively.

An example for ESME could be Voice Mail server, it has SMPP connection to the SMC from there can notify the subscriber about them mail box status by sending them SMS.

Therefore IN is also EMSE capable since you got balance notification in SMS format, a content provider such news agency server is also ESME.

2.2.2 SMS Flow

The Short Message Service is realized by the use of the Mobile Application Part (MAP) of the SS#7 protocol, with Short Message protocol elements being transported across the network as fields within the MAP messages. These MAP messages may be transported using 'traditional' TDM based signalling, or over IP using SIGTRAN and an appropriate adaptation layer. The Short Message protocol itself is defined by 3GPP TS 23.040 for the Short Message Service - Point to Point (SMS-PP), and 3GPP TS 23.041 for the Cell Broadcast Service (CBS).

Four MAP procedures are defined for the control of the Short Message Service:

Mobile Originated (MO), Mobile Terminated (MT), alert procedure,And waiting data set procedure.

I. Mobile Originated (MO)

The diagram to the right depicts a simplified call flow for a successful submission of a mobile originated short message.

When the subscriber sends a short message, the handset sends the text message over the air interface to the VMSC/SGSN. Along with the actual text of the short message, the destination address of the

SM and the address of the Short Message Service Centre (SMSC) are included, the latter taken from the handset's configuration stored on the SIM card.

Regardless of the air interface technology, the VMSC/SGSN invokes the MAP service package MAP_MO_FORWARD_SHORT_MESSAGE to send the text to the Interworking MSC of the Service Centre whose address was provided by the handset. This service sends the mo-ForwardSM. MAP operation to the SMSC identified in the SM Submission from the handset, embedded within a Transaction Capabilities Application Part (TCAP) message, and transported over the core network using the Signaling Connection Control Part (SCCP). The Interworking MSC of the SMSC, on receipt of the MAP mo-ForwardSM message, passes the SMS-PP Application Protocol Data Unit (APDU) containing the text message to the actual Service Centre (SC) of the SMSC for storing, and subsequent 'forwarding' (delivery) to the destination address and the SC returns an acknowledgement indicating success or failure. The message submission status is then forwarded, over the air interface, to the subscriber's handset.

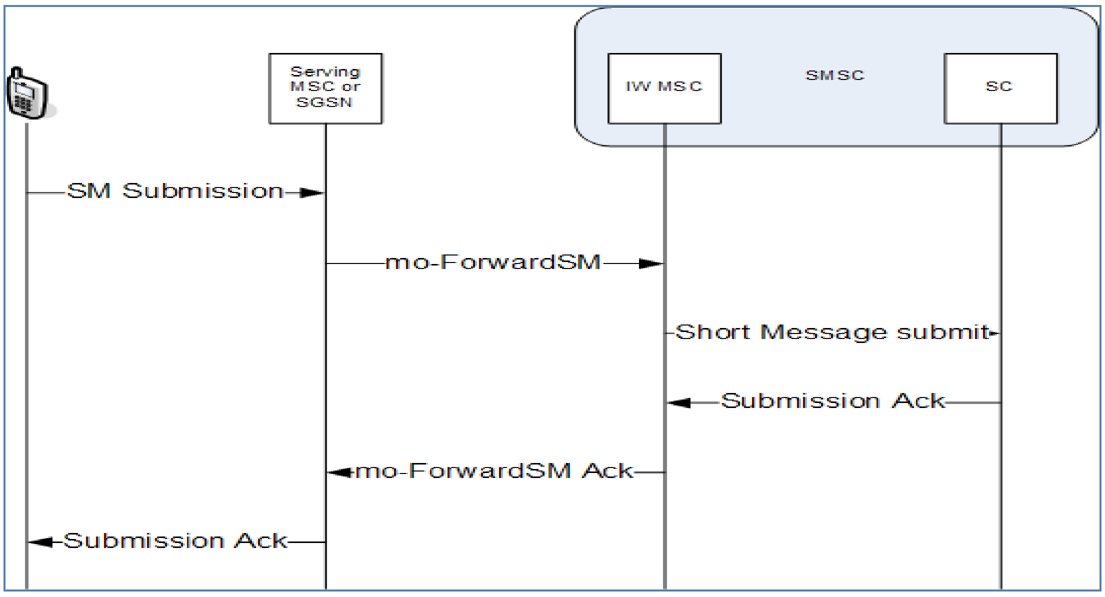


Figure2.6: MO call flow

II. Mobile Terminated (MT)

figure11 depicts a call flow for Mobile Terminated short message delivery. For the sake of simplicity, some of the interactions between the VMSC and VLR, and VMSC and Handset, have been omitted. When the SMSC determines it needs to attempt to deliver a short message to its destination, it will send the SMS-PP APDU containing the text message, the 'B-Party' (destination phone number) and other details to the Gateway MSC (GMSC) logical component on the SMSC. The GMSC, on receipt of this short message, needs to discover the location of the B-Party in order to be able to correctly deliver the text to the recipient (the term Gateway MSC, in this context, indicating an MSC that is obtaining routing information from the Home Location Register (HLR)). To do this, the GMSC invokes the MAP service package MAP_SEND_ROUTING_INFO_FOR_SM, which sends a sendRoutingInfoForSM (SRI-for-SM) MAP message to the destination number's HLR, requesting their present location. This

SRI-for-SM message may be sent to an HLR in the same network as the SMSC, or via an interconnect to an HLR in a foreign PLMN, depending on which network the destination subscriber belongs to.

The HLR performs a database lookup to retrieve the B-Party's current location, and returns it in an acknowledgement message to the SMSC's GMSC entity. The current location may be the MSC address the subscriber is currently roaming on, the SGSN address, or both. The HLR may also return a failure, if it considers the destination to be unavailable for short messaging; see the Failed short message delivery section below. Having obtained the routing information from the HLR, the GMSC will attempt to deliver the short message to its recipient. This is done by invoking the MAP_MT_FORWARD_SHORT_MESSAGE service, which sends a MAP mt-ForwardSMc message to the address returned by the HLR, regardless of whether it is an MSC (Circuit Switched SMS delivery) or an SGSN (Packet Switched SMS delivery).

The VMSC will request the information needed for it to deliver the Short Message to its recipient by sending a Send_Info_for_MT_SMS message to the VLR. The VLR will then instigate a page request, or subscriber search, for the destination subscriber's Mobile Subscriber ISDN Number (MSISDN), and return the result to the VMSC.

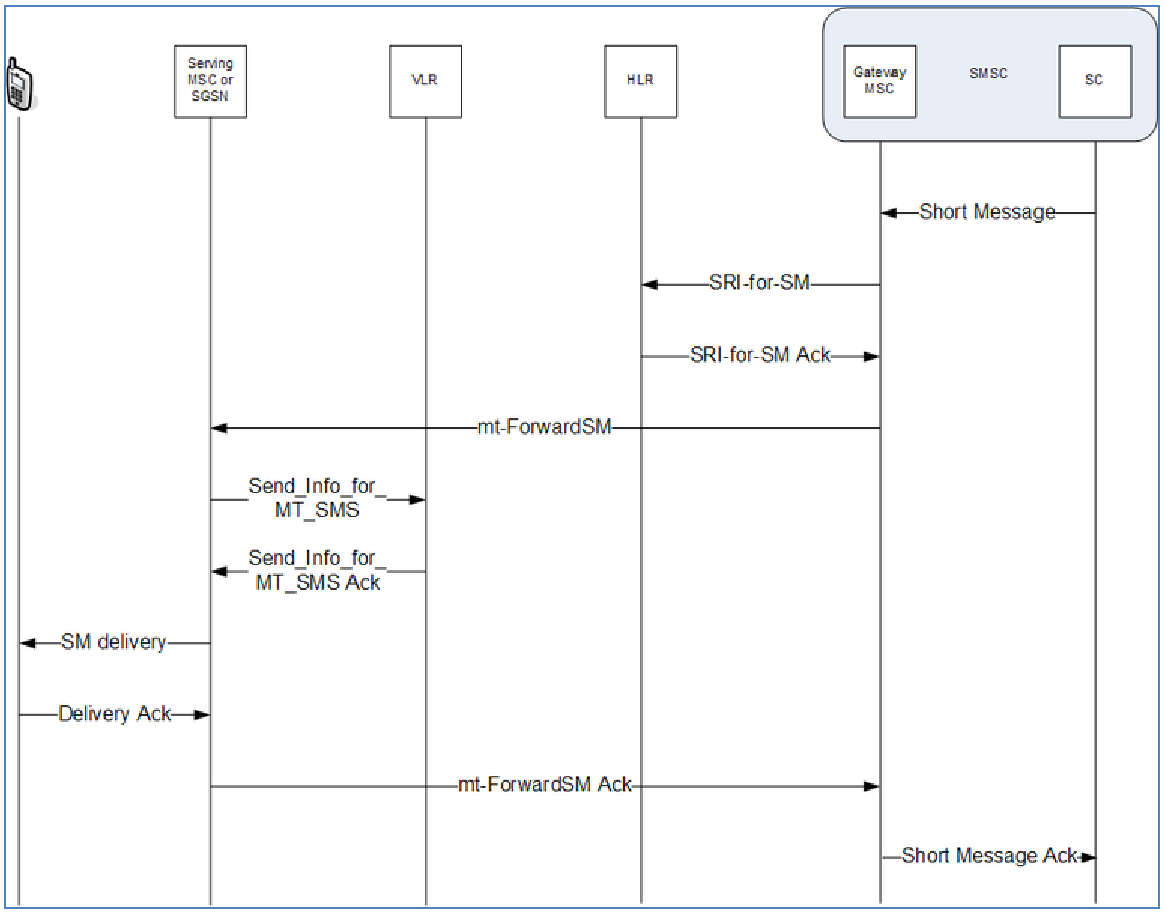


Figure 1.5.7:: MT call flow

Since a typical deployment sees the VLR being co-located with the MSC, this message flow is usually internal to the platform. Should the page or search for the subscriber fail, the VLR will indicate the failure cause to the VMSC which will abort the Short Message delivery procedure and return the failure to the SMSC (see the Failed short message delivery section below). If the page of the handset was successful, the VMSC will then send to the SMSC indicating successful delivery. The GMSC component of the SMSC passes the result of the delivery attempt to the Service Centre. In the case of successful delivery, the delivered text message will be removed from the Store and Forward Engine (SFE) and, if requested, a delivery report sent to the text originator. If the delivery failed, the SMSC invokes a retry procedure to periodically make further attempts at delivery; additionally, it may register with the HLR to receive a notification when the B-Party becomes available for short message delivery in the future (see the Failed short message delivery section below).

III. Failed short message delivery

When the VMSC/SGSN indicates a short message delivery failure, the SMSC may send a message to the HLR, using the MAP_REPORT_SM_DELIVERY_STATUS procedure, indicating the reason for the delivery failure and requesting that the SMSC be put on a list of service centres wanting to be notified when the destination party becomes available again. The HLR will set a flag against the destination account, indicating that it is unavailable for short message delivery, and store the SMSC's address in the Message Waiting Data (MWD) list for the destination party. Valid flags are Mobile Not Reachable Flag (MNRF), Memory Capacity Exceeded Flag (MCEF) and Mobile Not Reachable for GPRS (MNRG). The HLR will now start responding to SRI-for-SM requests with a failure, indicating the failure reason, and will automatically add the requesting SMSC's address to the MWD list for the destination party. (However if the SRI-for-SM message has priority flag set then the HLR will reply with VLR address if available) The HLR may be informed of a subscriber becoming available for short message delivery in several ways:

Where the subscriber has been detached from the network, a reattach will trigger a Location Update to the HLR.

Where the subscriber has been out of coverage, but not fully detached from the network, on coming back into coverage it will respond to page requests from the Visitor Location Register (VLR).

The VLR will then send a Ready-for-SM (mobile present) message to the HLR.

Where the MS has had its memory full, and the subscriber deletes some texts, a Ready-for-SM (memory available) message is sent from the VMSC/VLR to the HLR.

Upon receipt of an indication that the destination party is now ready to receive short messages, the HLR sends an AlertSC MAP message to each of the SMSCs registered in the MWD list for the subscriber, causing the SMSC to start the Short Message delivery process again, from the beginning.

Additionally, the SMSC will go into a retry schedule, attempting to periodically deliver the SM without getting an alert. The retry schedule interval will depend on the original failure cause - transient network failures will result in short retry schedule, whereas out of coverage will typically result in a longer schedule.

2.3 AT Commands

2.3.1 Hayes' commands

AT commands are used to control MODEMs. AT is the abbreviation for Attention. These commands come from Hayes commands that were used by the Hayes smart modems. The Hayes commands started with AT to indicate the attention from the MODEM. The dial up and wireless MODEMs (devices that involve machine to machine communication) need AT commands to interact with a computer. These include the Hayes command set as a subset, along with other extended AT commands.

Prior to the introduction of the Bulletin Board System (BBS), modems typically operated on direct-dial telephone lines that always began and ended with a known modem at each end. The modems operated in either "originate" or "answer" modes, manually switching between two sets of frequencies for data transfer. Generally the user placing the call would switch their modem to "originate" and then dial the number by hand. When the remote modem answered, already set to "answer" mode, the telephone handset was switched off and communications continued until the caller manually disconnected.

When automation was required, it was commonly only needed on the answer side - for instance, a bank might need to take calls from a number of branch offices for end-of-day processing. To fill this role, some modems included the ability to pick up the phone automatically when it was in answer mode, clearing the line when the other user manually disconnected. The need for automated outbound dialling was considerably less common, and handled through a separate peripheral device, a "dialler". This was normally plugged into a separate input/output port on the computer (typically an RS-232 port) and programmed separately from the modem itself.

This method of operation worked satisfactorily in the 1960s and early 1970s, when modems were generally used to connect dumb devices like computer terminals (dialling out) with smart mainframe computers (answering). However, the microcomputer revolution of the 1970s led to the introduction of low-cost modems and the idea of a semi-dedicated point-to-point link was no longer appropriate. There were potentially thousands of users who might want to dial any of the other thousands of users, and the only solution at the time was to make the user dial manually.

The computer industry needed a way to tell the modem what number to dial through software. The earlier separate dialers had this capability, but only at the cost of a separate port, which a microcomputer might not have available. Another solution would have been to use a separate set of "command pins" dedicated to sending and receiving commands, another could have used a signal pin indicating that the modem should interpret incoming data as a command. Both of these had hardware support in the RS-232 standard. However, many implementations of the RS-232 port on microcomputers were extremely basic, and some eliminated many of these pins as a cost saving measure.

2.3.2 Hayes' solution

Hayes Communications introduced a solution in its 1981 Smartmodem by re-using the existing data pins with no modification. Instead, the modem itself could switch itself between one of two modes:

1. data mode in which the modem sends the data to the remote modem. (A modem in data mode treats everything it receives from the computer as data and sends it across the phone line).
2. command mode in which data is interpreted as commands to the local modem (commands that the local modem should execute).

To switch from data mode to command mode, sessions sent an escape sequence string of three plus signs ("+++") followed by a pause of about a second. The pause at the end of the escape sequence was required to reduce the problem caused by in-band signaling: if any other data was received within one second of the three plus signs, it was not the escape sequence and would be sent as data. To switch back they sent the *online* command, O. In actual use many of the commands automatically switched to the online mode after completion, and it is rare for a user to use the online command explicitly.

In order to avoid licensing Hayes's patent, some manufacturers implemented the escape sequence without the time guard interval (TIES). This had a major denial of service security implication in that it would lead to the modem hanging up the connection should the computer ever try to transmit the byte sequence "+++ATH0" in data mode. For any computer connected to the Internet through such a modem, this could be easily exploited by sending it a ping of death request containing the sequence "+++ATH0" in the payload. The computer operating system would automatically try to reply the sender with the same payload, immediately disconnecting itself from the Internet, as the modem would interpret the ICMPpacket's data payload as a Hayes command.[1] The same error would also trigger if, for example, the user of the computer ever tried to send an e-mail containing the aforementioned string.

2.3.3 Commands

The Hayes command set includes commands for various phone-line manipulations, dialing and hanging-up for instance. It also includes various controls to set up the modem, including a set of register commands which allowed the user to directly set the various memory locations in the original Hayes modem. The command set was copied largely verbatim, including the meaning of the registers, by almost all early 300 baud modem manufacturers, of which there were quite a few.

The expansion to 1200 and 2400 baud required the addition of a small set of new commands, some of them prefixed with an ampersand ("&") to denote those dedicated to new functionality. Hayes itself was forced to quickly introduce a 2400 baud model shortly after their 1200, and the command sets were identical as a time-saving method.[2] Essentially by accident, this allowed users of existing 1200 baud modems to use the new Hayes 2400 models without changing their software. This re-inforced the use of the Hayes versions of these commands. Years later, the TIA/EIA raised the 2400-baud command set into a formal standard with the title Data Transmission Systems and Equipment - Serial Asynchronous Automatic Dialing and Control, TIA/EIA-602.

However Hayes Communications moved only slowly to higher speeds or the use of compression, and three other companies led the way here – Microcom, U.S. Robotics and Telebit. Each of these three used its own additional command-sets instead of waiting for Hayes to lead the way. By the early-1990s there were four major command sets in use, and a number of versions based on one of these. Things became simpler again during the widespread introduction of 14.4 and 28.8 kbit/s modems in the early 1990s. Slowly a set of commands based heavily on the original Hayes extended set using "&" commands became popular, and then universal. Only one other command set has remained popular, the US Robotics set from their popular line of modems.

The following text lists part of the Hayes command set (also called the AT commands: "AT" meaning attention).

The Hayes command set can subdivide into four groups:

I. basic command set - A capital character followed by a digit. For example, M1.
II. extended command set - An “&” (ampersand) and a capital character followed by a digit. This extends the basic command set. For example, &M1. Note that M1 is different from &M1.
III. proprietary command set - Usually starting either with a backslash (“\”) or with a percent sign (“%”); these commands vary widely among modem-manufacturers.
IV. register commands - Sr=n where r is the number of the register to be changed, and n is the new value that is assigned.

A register represents a specific physical location in memory. Modems have small amounts of memory on board. The fourth set of commands serves for entering values into a particular register (memory location). The register will store a particular variable (alpha-numeric information) which the modem and the communications software can utilize. For example, S7=60 instructs the computer to "Set register #7 to the value 60".

Although the command-set syntax defines most commands by a letter-number combination (L0, L1 etc.), the use of a zero is optional. In this example, "L0" equates to a plain "L". Keep this in mind when reading the table below.

When in data-mode an escape sequence can return the modem to command mode. The normal escape sequence is three plus signs

("+++"), and to disambiguate it from possible real data, a guard timer is used: it must be preceded by a pause, not have any pauses between the plus signs, and be followed by a pause; by default a "pause" is one second and "no pause" is anything less.

2.3.3.1 Syntactical definitions

The following syntactical definitions apply:

<CR> Carriage return character, is the command line and result code terminator character, which value, in decimal ASCII between 0 and 255, is specified within parameter S3. The default value is 13.

<LF> Linefeed character, is the character recognised as line feed character. Its value, in decimal ASCII between 0 and 255, is specified within parameter S4. The default value is 10. The line feed character is output after carriage return character if verbose result codes are used (V1 option used) otherwise, if numeric format result codes are used (V0 option used) it will not appear in the result codes.

<...> Name enclosed in angle brackets is a syntactical element. They do not appear in the command line.

[...] Optional subparameter of a command or an optional part of TA information response is enclosed in square brackets. Brackets themselves do not appear in the command line. When subparameter is not given in AT commands which have a Read command, new value equals its previous value. In AT commands which do not store the values of any of their subparameters, and so have not a Read command, which are called action type commands, action should be done on the basis of the recommended default setting of the subparameter.

Modem initialization[edit]

For other uses, see initialization vector.

A string can contain many Hayes commands placed together, so as to optimally prepare the modem to dial out or answer, e.g. AT&F&D2&C1S0=0X4. This is called the initialization string.[4] The V.250 specification requires all DCEs to accept a body (after "AT") of at least 40 characters of concatenated commands.[5]

Example session[edit]

The following represents two computers, computer A and computer B, both with modems attached, and the user controlling the modems with terminal-emulator software. Terminal-emulator software typically allows the user to send Hayes commands directly to the modem, and to see the responses. In this example, the user of computer A makes the modem dial the phone number of modem B at phone number 555-1234 (long distance). Note that after every command and response, there is a carriage return sent to complete the command.

Modem A	Modem B	Comment
ATDT15551234		User at modem A issues a dial command: AT-Get the modem's ATtention D-Dial T-Touch-Tone 15551234-Call this number

RING   Modem A begins dialing. Modem B's phone-line rings, and the modem reports the fact.

ATA   Computer at modem B issues answer command.

CONNECT   CONNECT The modems connect, and both modems report "connect". (In practice, most modems report more information after the word CONNECT — specifying the speed of the connection.) Also, at this time, both modems will raise the DCD, or Data Carrier Detect signal, on the serial port.

abcdef   abcdef   When the modems are connected, any characters typed at either side will appear on the other side. The person at computer A starts typing. The characters pass through the modem and appear on computer B's screen. (User A may not see his own typed characters — depending on the terminal software's local echo setting).

+++   The person at computer B issues the modem escape command. (Alternately, and more commonly, the computer B could drop the DTR, or Data Terminal Ready signal, to achieve a hangup, without needing to use +++ or ATH.)

OK   The modem acknowledges it.

ATH   The person at computer B issues a hang up command.

NO CARRIER OK   Both modems report that the connection has ended. Modem B responds "OK" as the expected result of the command; modem A says NO CARRIER to report that the remote side interrupted the connection. The modems on both sides drop their DCD signals as well.

2.3.3.2 Compatibility

While the original Hayes command set represented a huge leap forward in modem-based communications, with time many problems set in, almost none of them due to Hayes per se:

Due to the lack of a written standard, other modem manufacturers just copied the external visible commands and (roughly) the basic actions. This led to a wide variety of subtle differences in how modems changed from state to state, and how they handled error conditions, hangups, and timeouts.

Each manufacturer tended to add new commands to handle emerging needs, often incompatible with other modems.

For example, setting up hardware or software handshaking often required many different commands for different modems. This undermined the handy universality of the basic "AT" command-set.

Many "Hayes-compatible" modems had serious quirks that made them effectively incompatible. For example, many modems required a pause of several seconds after receiving the "AT Z" reset command. Some modems required spaces between commands, while others did not. Some would unhelpfully change baud-rate of their own "volition", which would leave the computer with no clue how to handle the incoming bits.

As a result of all this, eventually many communications programs had to give up any sense of being able to talk to all "Hayes-compatible" modems, and instead the programs had to try to determine the modem type from its responses, or provide the user with some option whereby they could enter whatever special commands it took to coerce their particular modem into acting properly.

2.3.3.3 The basic Hayes command set

The following commands are understood by virtually all modems supporting an AT command set, whether old or new.

Table 2.1: The basic Hayes command set

Command	**Description**	**Comments**
A0 or **A**	Answer incoming call	
A/	Repeat last command	Don't preface with **AT**, don't follow with carriage return. Enter usually aborts.
D	Dial	Dial the following number and then handshake P - Pulse Dial T - Touch Tone Dial W - Wait for the second dial tone R - Reverse to answer-mode after dialing @ - Wait for up to 30 seconds for one or more ringbacks , - Pause for the time specified in register S8 (usually 2 seconds) ; - Remain in command mode after dialing. ! - Flash switch-hook (Hang up for a half second, as in transferring a call.) L - Dial last number
E0 or **E**	No Echo	Will not echo commands to the computer
E1	Echo	Will echo commands to the computer (so one can see what one types)

H0	Hook Status	On hook. Hangs up the phone, ending any call in progress.
H1	Hook status	Off hook. Picks up the phone line (typically you'll hear a dialtone)
I0 to **I9**	Inquiry, Information, or Interrogation	This command returns information about the model, such as its firmware or brand name. Each number (0 to 9, and sometimes 10 and above) returns one line of modem-specific information, or the word ERROR if the line isn't defined. Today, Windows uses this for Plug-and-play detection of specific modem types.
L0 or **Ln**(n=1 to 3)	Speaker Loudness. Supported only by some modems, usually external ones. Modems lacking speakers, or with physical volume controls, or ones whose sound output is piped through the sound card will not support this command.	Off or low volume
M0 or **M**	Speaker off, completely silent during dialing	**M3** is also common, but different on many brands
M1		Speaker on until remote carrier detected (i.e. until the other modem is heard)
M2		Speaker always on (data sounds are heard after`CONNECT`)
O	Return Online	Returns the modem back to the normal connected state after being interrupted by the "+++" escape code.
Q0 or **Q**	Quiet Mode	Off - Displays result codes, user sees command responses (e.g. `OK`)
Q1	Quiet Mode	On - Result codes are suppressed, user does not see responses.
S*n*	Select current register Note that **S*n***, **?** and =*r* are actually three separate commands, and can be given in separate **AT** commands.	Select register *n* as the current register
S*n*?		Select register *n* as the current register, and query its value. Using **?** on its own will query whichever register was most recently selected.

S*n*=*r*		Select register *n* as the current register, and store *r*in it. Using **=*r*** on its own will store into whichever register was most recently selected.
V0 or **V**	Verbose	Numeric result codes
V1		English result codes (e.g. `CONNECT`, `BUSY`, `NO CARRIER` etc.)
X0 or **X**	Smartmodem	Hayes Smartmodem 300 compatible result codes
X1		Usually adds connection speed to basic result codes (e.g. `CONNECT 1200`)
X2		Usually adds dial tone detection (preventing blind dial, and sometimes preventing **ATO**)
X3		Usually adds busy signal detection.
X4		Usually adds both busy signal and dial tone detection
Z0 or **Z**	Reset	Reset modem to stored configuration. Use **Z0**,**Z1**etc. for multiple profiles. This is the same as **&F**for factory default on modems without NVRAM (non volatile memory)

2.3.3.4 Modem S register definitions

Table 02.2: Modem S register definitions

Register	Description	Range	Default value
S0	Number of rings before Auto-Answer	0–0 never	0
S1	Ring Counter	0–255 rings	0

S2	Escape character	0–255, ASCII decimal	43 ("+")
S3	Carriage Return Character	0–127, ASCII decimal	13 (Carriage Return)
S4	Line Feed Character	0–127, ASCII decimal	10 (Line Feed)
S5	Backspace Character	0–32, ASCII decimal	8 (Backspace)
S6	Wait Time before Blind Dialing	2–255 seconds	2
S7	Wait for Carrier after Dial	1–255 seconds	50
S8	Pause Time for Comma (Dial Delay)	0–255 seconds	2
S9	Carrier Detect Response Time	1–255 tenths of a seconds	6 (0.6 second)
S10	Delay between Loss of Carrier and Hang-Up	1–255 tenths of a second	14 (1.4 seconds)
S11	DTMF Tone Duration	50–255 milliseconds	95 milliseconds
S12	Escape Code Guard Time	0–255 fiftieths of a second	50 (1 second)
S18	Test Timer	0–255 seconds	0 seconds
S25	Delay to DTR	0–255 (seconds if synchronous mode, hundredths of a second in all other	5

		modes)	
S26	RTS to CTS Delay Interval	0–255 hundredths of a second	1 hundredth of a second
S30	Inactivity Disconnect Timer	0–255 tens of seconds	0 (disable)
S37	Desired Telco Line Speed	0–10 Command options: • 0 Attempt auto mode connection • 1 Attempt to connect at 300 bit/s • 2 Attempt to connect at 300 bit/s • 3 Attempt to connect at 300 bit/s • 5 Attempt to connect at 1200 bit/s • 6 Attempt to connect at 2400 bit/s • 7 Attempt to connect in V.23 75/1200 mode. • 8 Attempt to connect at 9600 bit/s • 9 Attempt to connect at 12000 bit/s • 10 Attempt to connect at 14400 bit/s	0
S38	Delay before Force Disconnect	0–255 seconds	20 seconds

2.3.3.5 V.250

The ITU-T established a standard in its V-Series Recommendations, V.25 ter, in 1995 in an attempt to establish a standard for the command set again. It was renamed V.250 in 1998 with an annex that was not concerning the Hayes command set renamed as V.251. A V.250 compliant modem implements the A, D, E, H, I, L, M, N, O, P, Q, T, V, X, Z, &C, &D, and &F commands in the way specified by the standard. It must also implement S registers and must use registers S0, S3, S4, S5, S6, S7, S8, and S10 for the purposes given in the standard. Lastly it also must implement any command beginning with the plus sign, "+" followed by any letter A to Z, only in accordance with ITU recommendations. Modem manufacturers are free to implement other commands and S-registers as they see fit, and may add options to standard commands.

V.250 – Defined leading character sequences

Leading characters	Includes commands related to
+A	Call control (network Addressing) issues, common, PSTN, ISDN,
+C	Digital Cellular extensions
+D	Data Compression, ITU-T Rec. V.42 bis
+E	Error Control, ITU-T Rec. V.42
+F	Facsimile, ITU-T Rec. T.30, etc.
+G	Generic issues such as identity and capabilities

+I	DTE-DCE Interface issues, ITU-T Rec. V.24, etc.
+M	Modulation, ITU-T Rec. V.32 bis, etc.
+P	PCM DCE commands, ITU-T Rec. V.92
+S	Switched or Simultaneous Data Types
+T	Test issues
+V	Voice extensions
+W	Wireless extensions

2.3.3.6 GSM

The ETSI GSM 07.07 (3GPP TS 27.007) specifies AT style commands for controlling a GSM phone or modem. The ETSI GSM 07.05 (3GPP TS 27.005) specifies AT style commands for managing the SMS feature of GSM.

Examples of GSM commands:

Table 2.3: GSM commands

Command	**Description**
AT+CPIN=1234	Enter PIN code
AT+CPWD="SC","old","new"	Change PIN code from 'old' to 'new'
AT+CLCK="SC",0,"1234"	Remove PIN code
AT&V	Status

ATI	Status (Manufacturer, Model, Revision, IMEI, capabilities)
AT+COPS=?	List available networks 0-Unknown/2-Current/3-Forbidden, Longname, Shortname, Numerical-ID, "AcT"
AT+CSQ	Get signal strength. Answer: +CSQ: <rssi (more=better)>, <ber, less=better>
ATD*99#	Dial access point
AT+CGDCONT=1,"IP","access.point.name"	Defines PDP context[6]

GSM/3G modems typically support the ETSI GSM 07.07/3GPP TS 27.007 AT command set extensions, although how many commands are implemented varies.

Most USB modem vendors, such as Huawei, Sierra Wireless, Option, have also defined proprietary extensions for radio mode selection (GSM/3G preference) or similar. Some recent high speed modems provide a virtual Ethernet interface instead of using a PPP connection for the data connection because of performance reasons (PPP connection is only used between the computer and the modem, not over network). The set-up requires vendor-specific AT command extensions. Sometimes the specifications for these extensions are openly available, other times the vendor requires an NDA for access to these.

3. Managing SMS using AT COMMANDS

3.1 Interfacing Principle

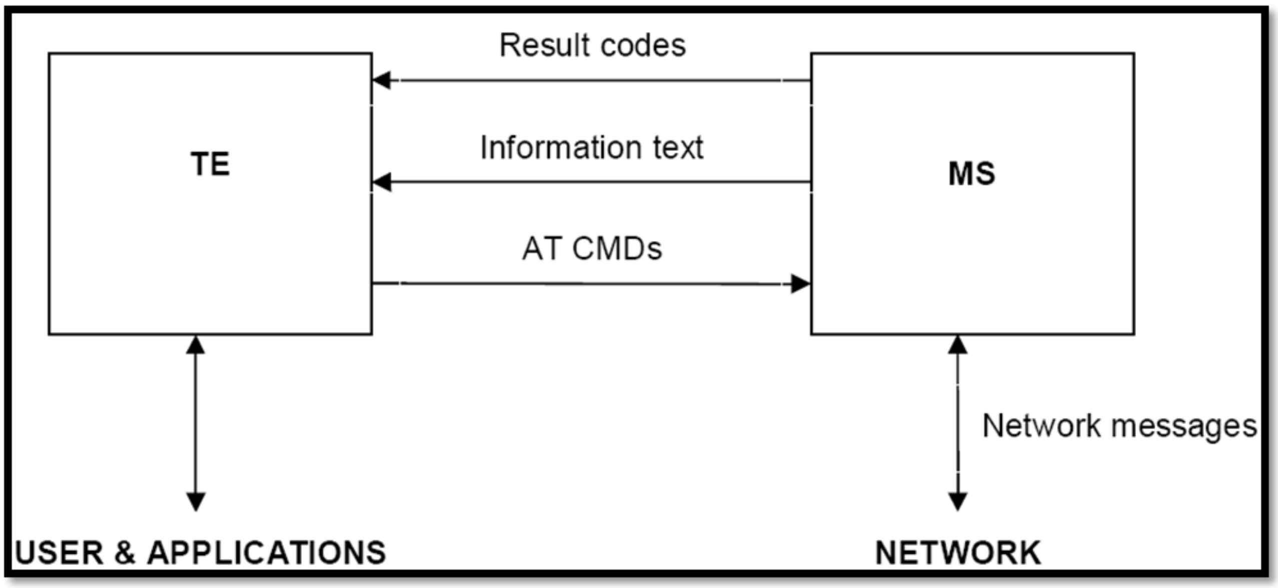


FIGURE 3.1: Block diagram of interaction between TE and MS

Since the AT command is a packet transmitted via communication port, the packet
size is limited. For the transmission of AT command, in addition to the two characters "AT", a maximum of 260 characters can be received (including the empty characters at the end). For the "response" message or URC reported by the board, the maximum length is limited to 668 characters.

Each command line can include only one AT command. For the URC instruction or response reported from MS to TE, only one AT command is allowed in a command line.

In order to make the commands and response formats more readable and standard, except the original interfaces of Qualcomm, in all newly added interfaces, e.g. no space can be contained in the commands such as AT^XXX: <arg0>, <arg1>, or behind ^, colon or comma. No redundant space is allowed at the head or end.

After delivering each AT command, the TE cannot deliver the second AT command until the MS has made response to this AT command. Otherwise, the
second AT command will not be executed.

For the AT command to which the response is given only after a long time, in order to prevent interference on other events, it is recommended to report the final execution result asynchronously. If the MS responds to the TE only after a long time of waiting, e.g. the "AT+CCFC=?" command receives a response only after a long time after the command is delivered, the MS may have received the reported instruction of RING on this occasion. Namely, the reporting of RING may interrupt other responses, and other URCs will not interrupt the response of command, and the interrupted part of the response will continue being reported.

Unless otherwise specified, all default codes between TE and MS take on this
format: GSM 7 bit Default Alphabet. See also Section 6 in protocol 23.038. The
character @ is transmitted on the interface still according to 0x00 of 7bit coding. The board software and API should be able to process this character. The board uploads the carriage return character (<CR>) and linefeed character (<LF>) in the string in the form of space.

A sort of compounding between quotation and comma cannot exist in the string in this current version. For the data format of UCS2 code, the code value should be
reported in the string format (if the code value is 0x553a, 553a should be reported).

The "Possible response" sent from MS to TE is composed of "Information text"
and "Result code", where "Information text" is optional, and "Result code" is mandatory.

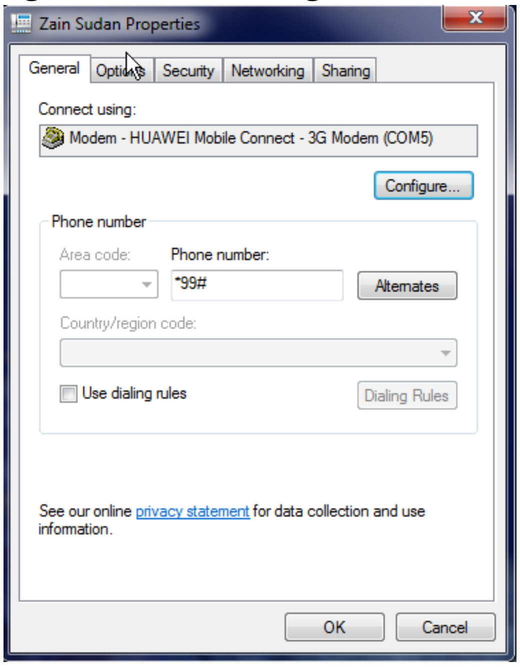
3.2 Summary of commands:

Table 3.1: Summary of commands

Command	Meaning
AT	*Attention (check)*
ATI	*Manufacturer identification*
AT+CIMI	*IMSI*
AT+CPIN?	*Check PIN*
AT+CSCA?	*Check SMC address*
AT+CSCA=	*Set SMC address*
at+cmgf=1	*Message Mode*
at+cmgs=	*Sending "Set destination address"*
[Ctrl+z]	*End Of message and execute sending*
AT+CPMS?	*Check memory*
at+cmgl="all"	*List All memory contents*
at=cmgr="N"	*Reading message in position N in the memory*

3.3 Case #1: Managing SMS via HyperTerminal

3.3.1 Checking the modem configuration


Zain Sudan Properties

General | Options | Security | Networking | Sharing

Connect using:

Modem - HUAWEI Mobile Connect - 3G Modem (COM5)

Configure...

Phone number

Area code:

Phone number: *99#

Alternates

Country/region code:

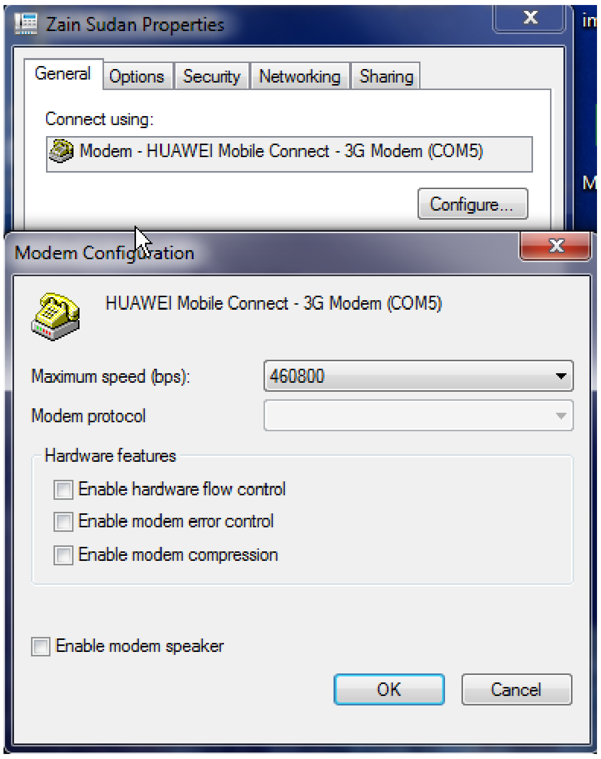
Use dialing rules

Dialing Rules

See our online privacy statement for data collection and use information.

OK | Cancel

Figure 3.2: Checking the modem configuration


Zain Sudan Properties

General | Options | Security | Networking | Sharing

Connect using:

Modem - HUAWEI Mobile Connect - 3G Modem (COM5)

Configure...

Modem Configuration

HUAWEI Mobile Connect - 3G Modem (COM5)

Maximum speed (bps): 460800

Modem protocol

Hardware features

Enable hardware flow control

Enable modem error control

Enable modem compression

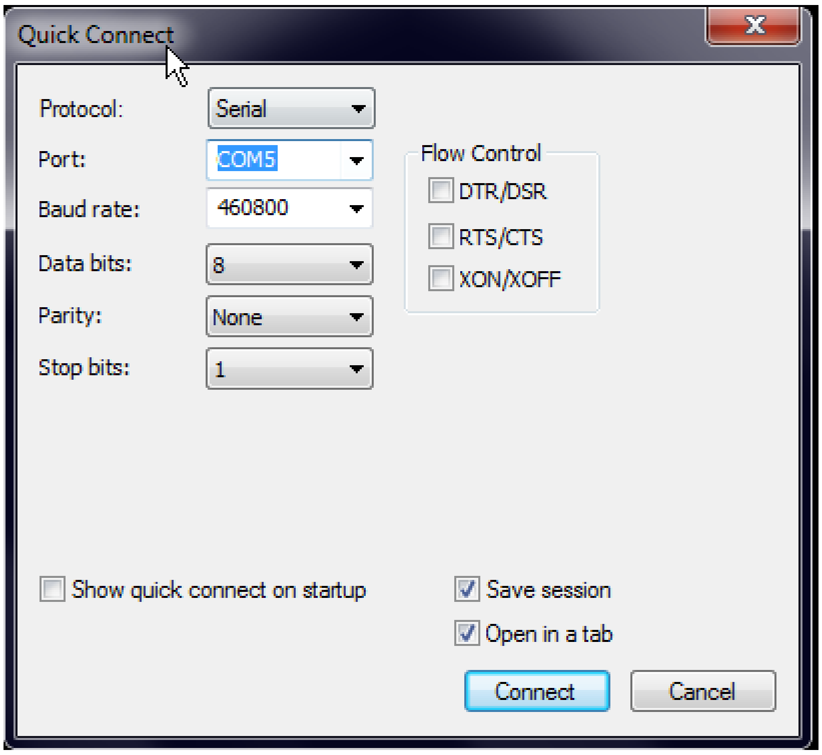
Enable modem speaker

OK | Cancel

Figure 3.3: Checking the modem Speed

3.3.2 Connecting using the Hyper Terminal

Applying the port settings which you got from the previous step to connect.



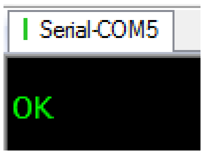
Protocol: Serial
Port: COM5
Baud rate: 460800
Data bits: 8
Parity: None
Stop bits: 1
Flow Control
DTR/DSR
RTS/CTS
XON/XOFF

Figure 3.4: Setup Hyper Terminal

Start sending the commands and wait for response of each command before sending the next command.

I AT

Command: AT

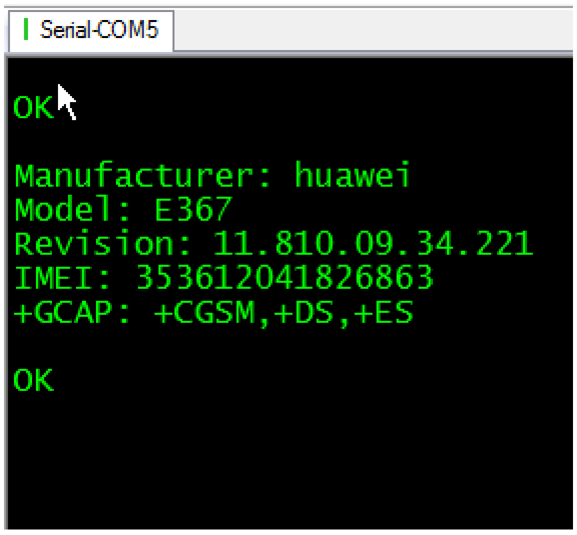


```
OK
```

Figure 3.5: AT cmd result

II Manufacturer identification

Command: ATI



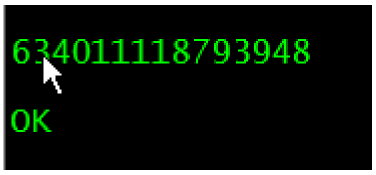
Serial-COM5

```
OK

Manufacturer: huawei
Model: E367
Revision: 11.810.09.34.221
IMEI: 353612041826863
+GCAP: +CGSM,+DS,+ES

OK
```

Figure 3.6: ATI cmd result

III IMSI

Command: AT+CIMI

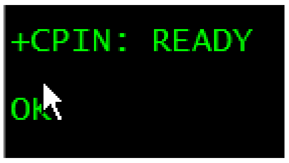


```
634011118793948

OK
```

Figure 3.7: AT+CIMI cmd result

IV PIN

Command: AT+CPIN?

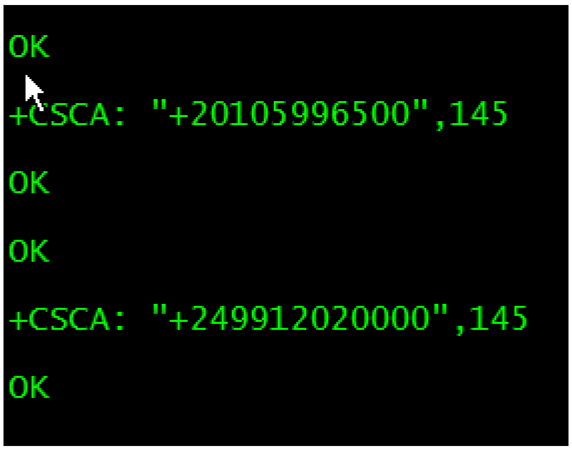


```
+CPIN: READY

OK
```

Figure 3.8: AT+CPIN? Cmd result

V SMC address

Command: AT+CSCA? , AT+CSCA=”+249912020000”



```
OK
+CSCA: "+20105996500",145
OK
OK
+CSCA: "+249912020000",145
OK
```

Figure 3.9: AT+ CSCA? Cmd result

VI Message Mode
at+cmgf=1 (send SMS in text mode)

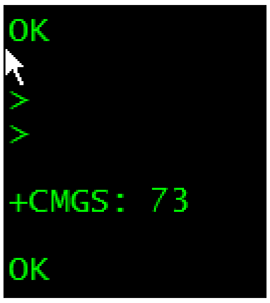
OK

VII Sending
at+cmgs=0912xxxxx

>

hello

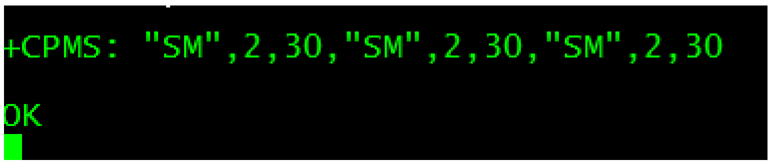
[Ctrl+z]



```
OK
>
>
+CMGS: 73
OK
```

Fig 3.10: Sending cmd result

VIII Check SMS memory
Command: AT+CPMS?

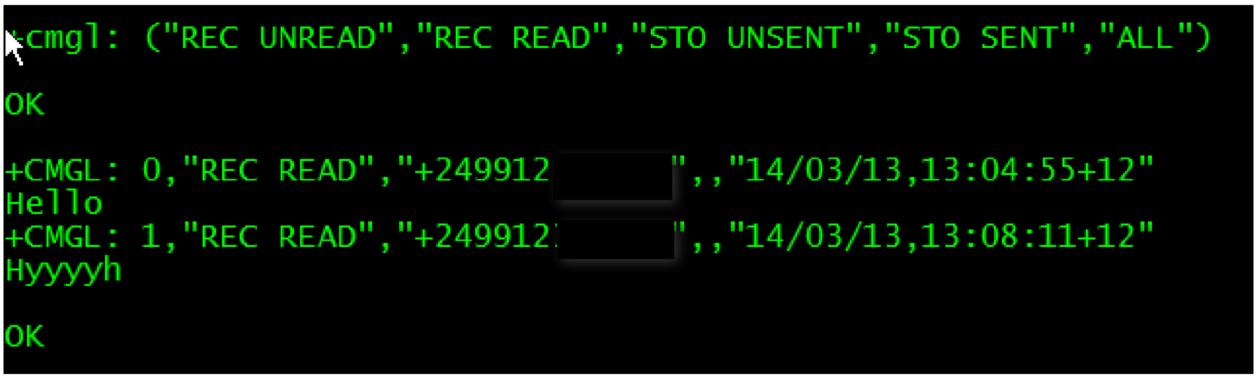


```
+CPMS: "SM",2,30,"SM",2,30,"SM",2,30

OK
```

Fig 3.11: AT+CPMS? cmd result

IX List Memory contents

Command: at+cmgl=? , at+cmgl="all"

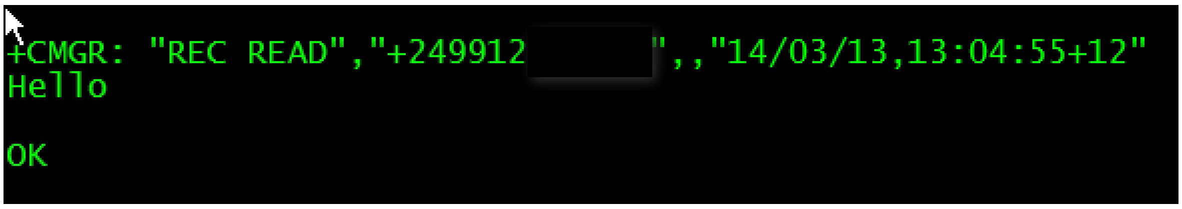


```
+cmgl: ("REC UNREAD","REC READ","STO UNSENT","STO SENT","ALL")

OK

+CMGL: 0,"REC READ","+249912[redacted]",,"14/03/13,13:04:55+12"
Hello
+CMGL: 1,"REC READ","+249912[redacted]",,"14/03/13,13:08:11+12"
Hyyyyh

OK
```

Fig 3.12: AT+CMGL=”ALL” cmd result

X Reading

Command: at=cmgr=”0”



```
+CMGR: "REC READ","+249912[redacted]",,"14/03/13,13:04:55+12"
Hello

OK
```

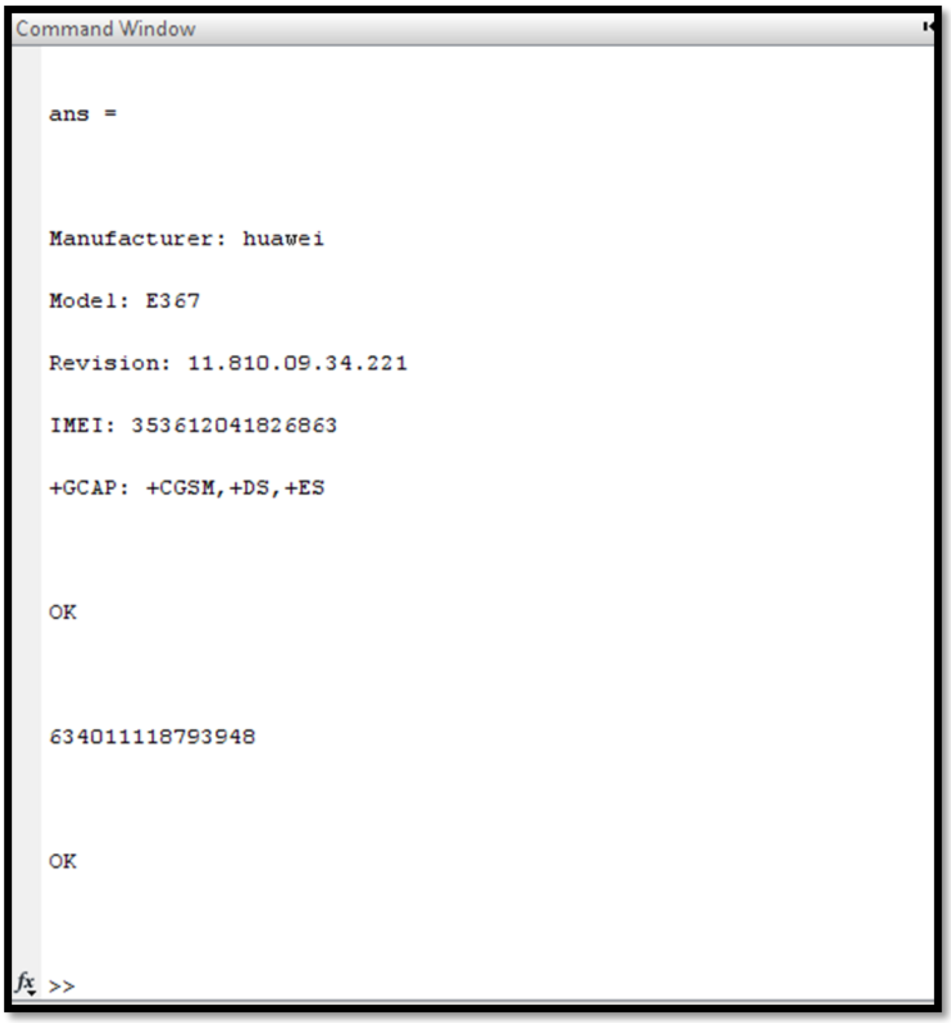
Fig 3.13: AT+CMGL=”ALL” cmd result

3.4 Case#2: Managing SMS via Matlab

3.4.1 Getting INFO

```
clc;
clear all;
global BytesAvail;
global A;
global B;
tx ='ATI';
tx1=char(13);
```

```
tx5='AT+CMGF=1';
tx2='AT+CIMI ';
s = serial('COM5');
s.baudrate=9600;
fopen(s);
s.Terminator = 'CR';
fprintf(s,'%s', tx);
fprintf(s,'%s', tx1);
BytesAvail=s.BytesAvailable;
    if(BytesAvail > 0), A=fread(s,BytesAvail,'char'); end
A;
sprintf('%c', A)
fprintf(s,'%s', tx2);
fprintf(s,'%s', tx1);
BytesAvail=s.BytesAvailable;
    if(BytesAvail > 0), B=fread(s,BytesAvail,'char'); end
B;
sprintf('%c', B)

fclose(s)
```

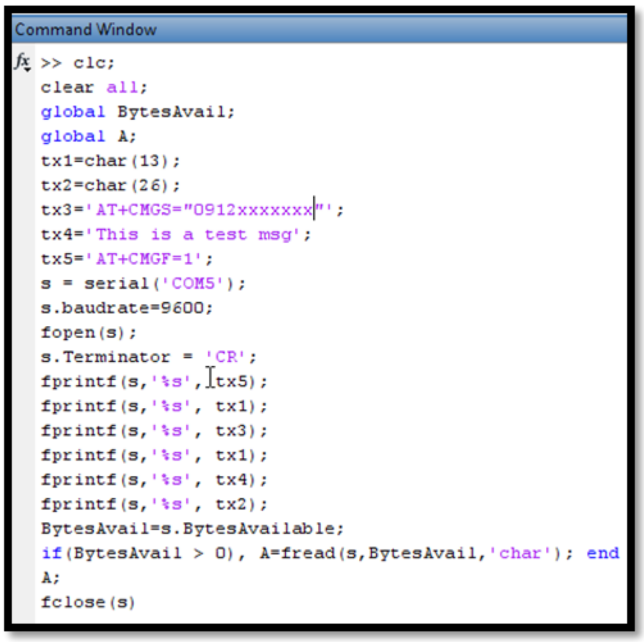


```
ans =


Manufacturer: huawei

Model: E367

Revision: 11.810.09.34.221

IMEI: 353612041826863

+GCAP: +CGSM,+DS,+ES


OK


634011118793948


OK

>>
```

Fig 3.14: MATLAB COMMAND WINDOW

3.4.2 Sending

```
clc;
clear all;
global BytesAvail;
global A;
tx1=char(13);
tx2=char(26);
tx3='AT+CMGS="0912xxxxxx"';
tx4='This is a test msg';
tx5='AT+CMGF=1';
s = serial('COM5');
s.baudrate=9600;
fopen(s);
s.Terminator = 'CR';
fprintf(s,'%s', tx5);
fprintf(s,'%s', tx1);
fprintf(s,'%s', tx3);
fprintf(s,'%s', tx1);
fprintf(s,'%s', tx4);
fprintf(s,'%s', tx2);
BytesAvail=s.BytesAvailable;
    if(BytesAvail > 0), A=fread(s,BytesAvail,'char'); end
A;
fclose(s)
```

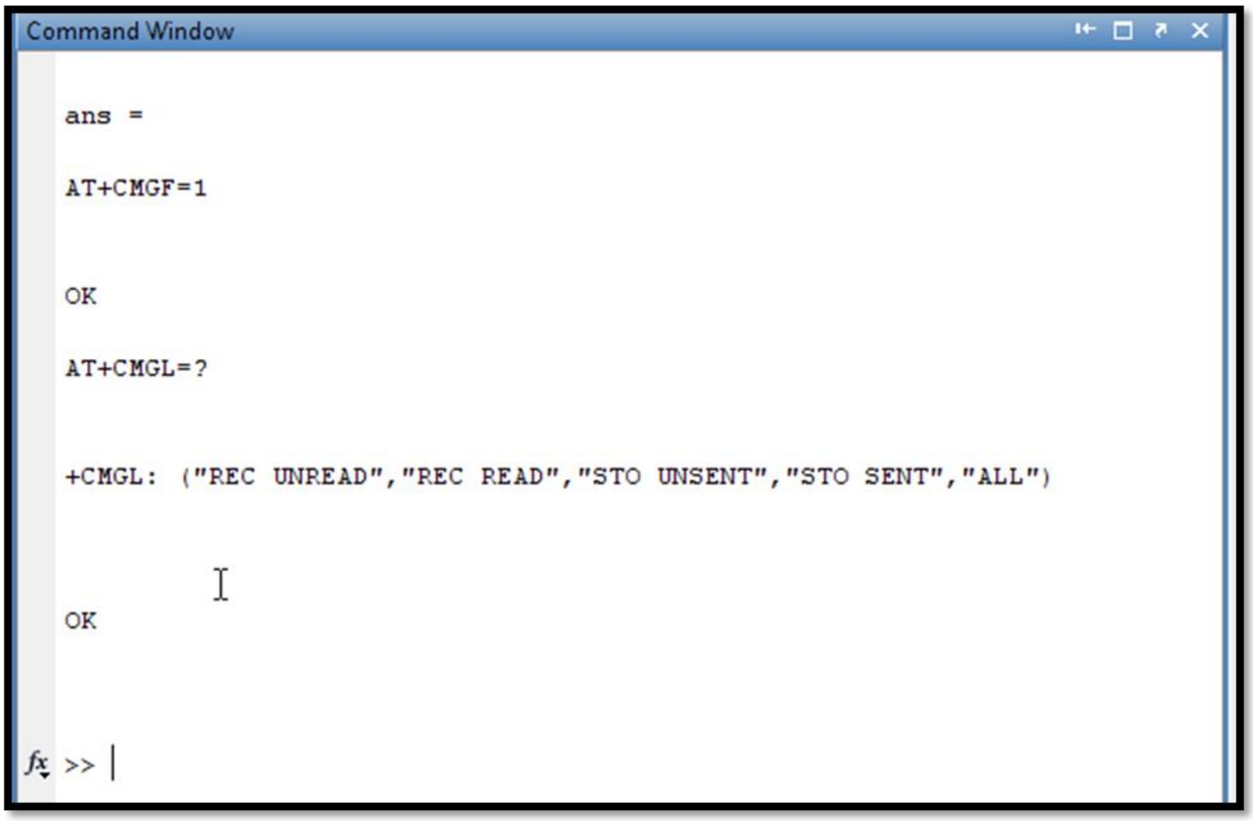


```
>> clc;
clear all;
global BytesAvail;
global A;
tx1=char(13);
tx2=char(26);
tx3='AT+CMGS="0912xxxxxxx"';
tx4='This is a test msg';
tx5='AT+CMGF=1';
s = serial('COM5');
s.baudrate=9600;
fopen(s);
s.Terminator = 'CR';
fprintf(s,'%s',tx5);
fprintf(s,'%s', tx1);
fprintf(s,'%s', tx3);
fprintf(s,'%s', tx1);
fprintf(s,'%s', tx4);
fprintf(s,'%s', tx2);
BytesAvail=s.BytesAvailable;
if(BytesAvail > 0), A=fread(s,BytesAvail,'char'); end
A;
fclose(s)
```

Fig 3.15: MATLAB COMMAND WINDOW

3.4.3 Listing

```
clc;
clear all;
global BytesAvail;
global A;
tx='AT+CMGL=?';
tx1=char(13);
tx5='AT+CMGF=1';
s = serial('COM5');
s.baudrate=9600;
fopen(s);
s.Terminator = 'CR';
fprintf(s,'%s', tx5);
fprintf(s,'%s', tx1);
fprintf(s,'%s', tx);
fprintf(s,'%s', tx1);
BytesAvail=s.BytesAvailable;
if(BytesAvail > 0), A=fread(s,BytesAvail,'char'); end
A;
sprintf('%c', A)
fclose(s)
```

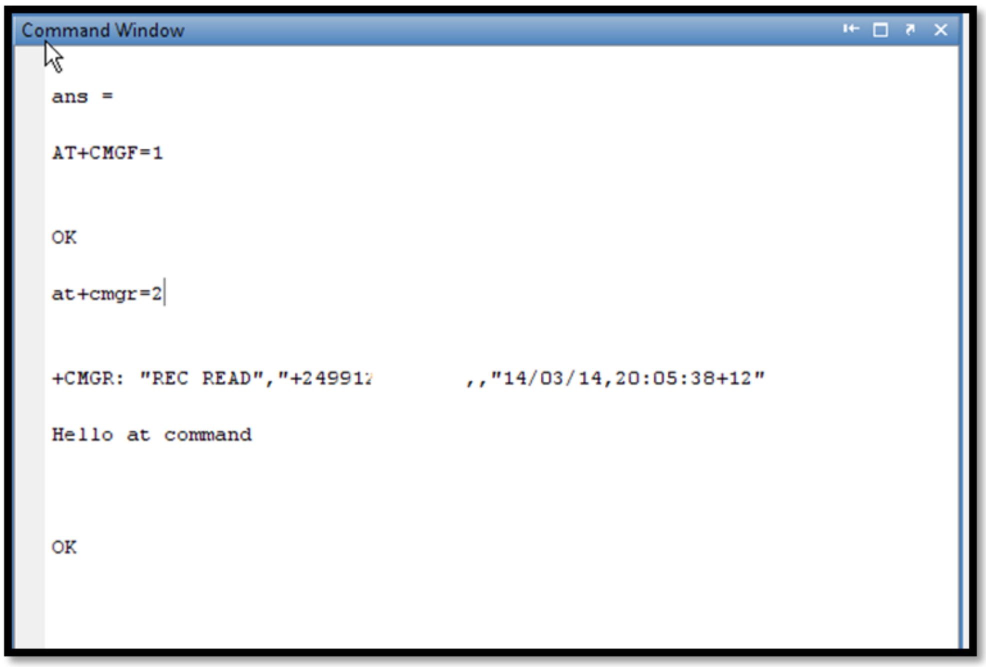


```
ans =

AT+CMGF=1


OK

AT+CMGL=?


+CMGL: ("REC UNREAD","REC READ","STO UNSENT","STO SENT","ALL")



OK



>>
```

Fig 3.16: MATLAB COMMAND CMGL result

3.4.4 Reading

```
clc;
clear all;
global BytesAvail;
global A;
tx ='at+cmgr=2';
tx1=char(13);
tx5='AT+CMGF=1';

s = serial('COM5'); % You have to replace this with your
3G modem's COMport number
s.baudrate=9600;
fopen(s);
s.Terminator = 'CR';
fprintf(s,'%s', tx5);
fprintf(s,'%s', tx1);
fprintf(s,'%s', tx);
fprintf(s,'%s', tx1);
BytesAvail=s.BytesAvailable;
    if(BytesAvail > 0), A=fread(s,BytesAvail,'char'); end
A;
sprintf('%c', A)
fclose(s)
```

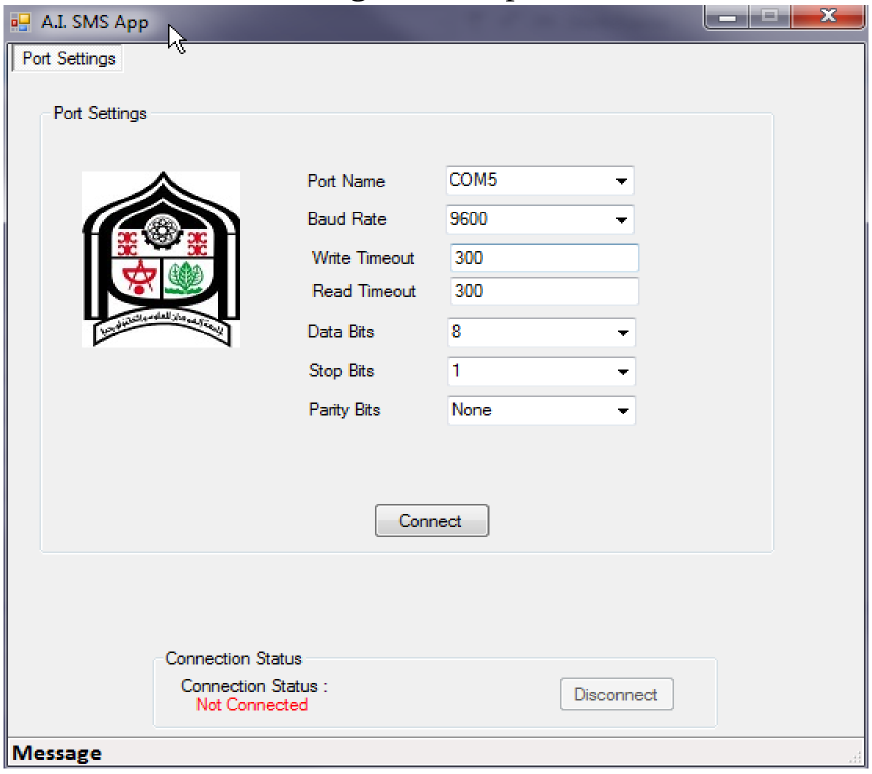


```
ans =

AT+CMGF=1


OK

at+cmgr=2


+CMGR: "REC READ","+24991:        ,,"14/03/14,20:05:38+12"

Hello at command



OK
```

Fig 3.17: MATLAB COMMAND CMGR result

3.5 Case#3: Managing SMS via C#

3.5.1 Connecting to serial port



Port Settings

Port Name COM5
Baud Rate 9600
Write Timeout 300
Read Timeout 300
Data Bits 8
Stop Bits 1
Parity Bits None

Connection Status
Connection Status :
Not Connected

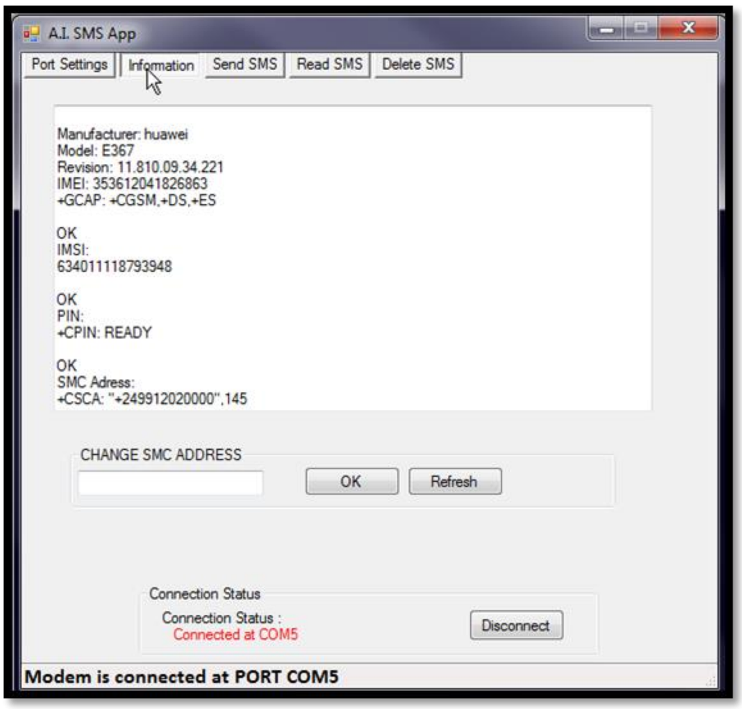
Message

Fig 3.18: (AI SMS APP – connecting to serial port)

```
  private void btnOK_Click(object sender, EventArgs e)
        {
            try
            {
                //Open communication port
                this.port =
objclsSMS.OpenPort(this.cboPortName.Text,
Convert.ToInt32(this.cboBaudRate.Text),
Convert.ToInt32(this.cboDataBits.Text),
Convert.ToInt32(this.txtReadTimeOut.Text),
Convert.ToInt32(this.txtWriteTimeOut.Text));

                if (this.port != null)
                {
this.gboPortSettings.Enabled = false;

//MessageBox.Show("Modem is connected at PORT " +
this.cboPortName.Text);
this.statusBar1.Text = "Modem is connected at PORT " +
this.cboPortName.Text;

this.btnDisconnect.Enabled = true;
                }
                else
                {
```

```
//MessageBox.Show("Invalid port settings");
this.statusBar1.Text = "Invalid port settings";
                    }
                }
                catch (Exception ex)
                {
                    ErrorLog(ex.Message);
                }}
```

3.5.2 Getting Info



```
Manufacturer: huawei
Model: E367
Revision: 11.810.09.34.221
IMEI: 353612041826863
+GCAP: +CGSM,+DS,+ES

OK
IMSI:
634011118793948

OK
PIN:
+CPIN: READY

OK
SMC Adress:
+CSCA: "+249912020000",145
```

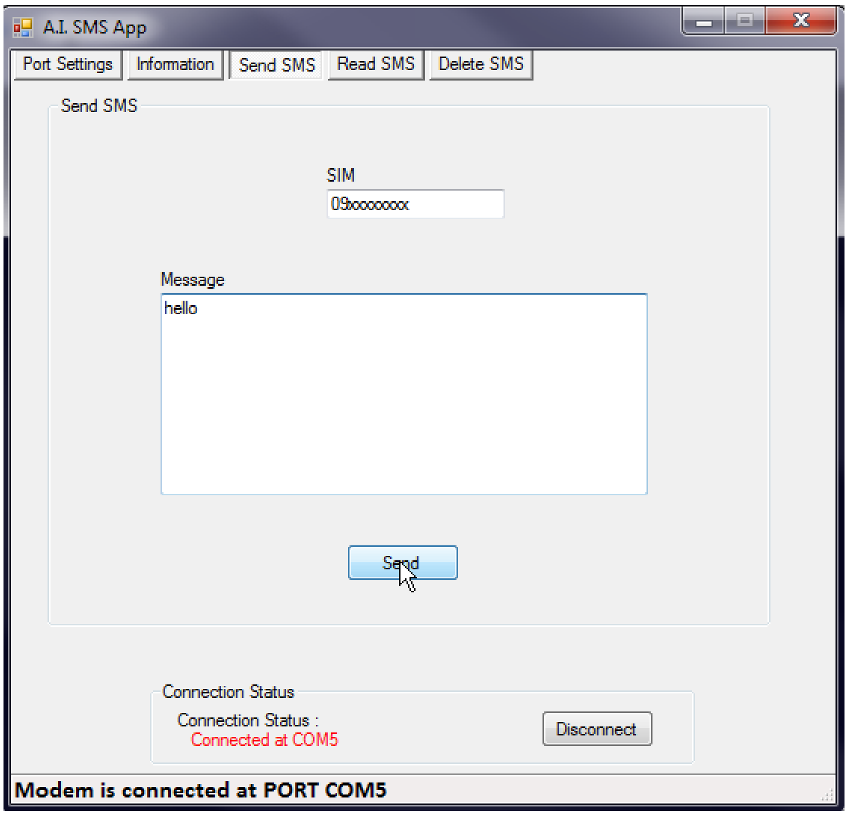
Connection Status :
Connected at COM5

Modem is connected at PORT COM5

Fig 3.19: (AI SMS APP – Getting info)

```
private void tabSMSapplication_Selected(object sender,
TabControlEventArgs e)
        {
                string input = objclsSMS.ExecCommand(port,
"ATI", 5000, "Failed to execute command");
                this.textBox1.Text = input;

                input = objclsSMS.ExecCommand(port,
"AT+CIMI", 5000, "Failed to execute command");
                this.textBox1.Text += "IMSI:" + input;
                input = objclsSMS.ExecCommand(port,
"AT+CPIN?", 5000, "Failed to execute command");
                this.textBox1.Text += "PIN:" + input;
                input = objclsSMS.ExecCommand(port,
"AT+CSCA?", 5000, "Failed to execute command");
                this.textBox1.Text += "SMC Adress:" + input;
                     }
```

3.5.3 Sending

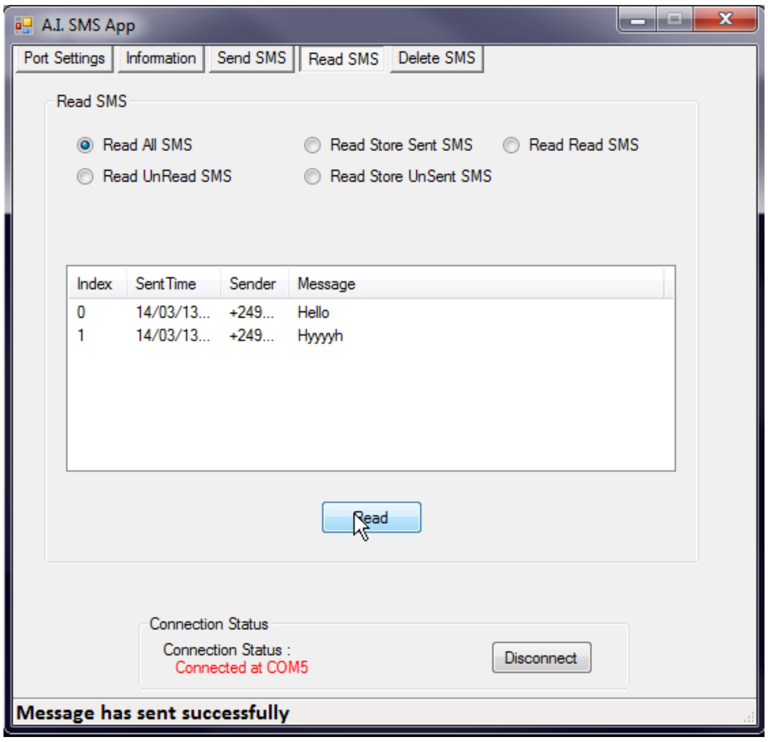


SIM: 09xxxxxxxxx
Message: hello
Connection Status : Connected at COM5
Modem is connected at PORT COM5

Fig 3.20: (AI SMS APP – Sending)

```
    private void btnSendSMS_Click(object sender, EventArgs
e)
        {


//........................................... Send SMS
....................................................
            try
            {

                if (objclsSMS.sendMsg(this.port,
this.txtSIM.Text, this.txtMessage.Text))
                {
//MessageBox.Show("Message has sent successfully");
this.statusBar1.Text = "Message has sent successfully";
                }
                else
                {
//MessageBox.Show("Failed to send message");
this.statusBar1.Text = "Failed to send message";
                }

            }
            catch (Exception ex)
            {
                ErrorLog(ex.Message);
            }
        }
```

3.5.3 Reading



(●) Read All SMS ( ) Read Store Sent SMS ( ) Read Read SMS
( ) Read UnRead SMS ( ) Read Store UnSent SMS

Index	Sent Time	Sender	Message
0	14/03/13...	+249...	Hello
1	14/03/13...	+249...	Hyyyyh

Connection Status
Connection Status :
Connected at COM5

Message has sent successfully

Fig 3.21: (AI SMS APP – Reading)

```
private void btnReadSMS_Click(object sender, EventArgs e)
        {
            lvwMessages.Items.Clear();
            try
            {
//count SMS
int uCountSMS = objclsSMS.CountSMSmessages(this.port);
if (uCountSMS > 0)
{

#region Command
string strCommand = "AT+CMGL=\"ALL\"";

if (this.rbReadAll.Checked)
{
strCommand = "AT+CMGL=\"ALL\"";
}
else if (this.rbReadUnRead.Checked)
{
strCommand = "AT+CMGL=\"REC UNREAD\"";
}
else if (this.rbReadStoreSent.Checked)
{
strCommand = "AT+CMGL=\"STO SENT\"";
}
else if (this.rbReadStoreUnSent.Checked)
{
strCommand = "AT+CMGL=\"STO UNSENT\"";
}
else if (this.ReadReadSMS.Checked)
{
strCommand = "AT+CMGL=\"REC READ\"";
}
#endregion

// If SMS exist then read SMS
#region Read SMS

objShortMessageCollection = objclsSMS.ReadSMS(this.port,
strCommand);
foreach (ShortMessage msg in objShortMessageCollection)
{

ListViewItem item = new ListViewItem(new string[] {
msg.Index, msg.Sent, msg.Sender, msg.Message });
item.Tag = msg;
lvwMessages.Items.Add(item);

}
#endregion

                }
                else
                {
lvwMessages.Clear();
//MessageBox.Show("There is no message in SIM");
this.statusBar1.Text = "There is no message in SIM";

                }
            }
            catch (Exception ex)
            {
                ErrorLog(ex.Message); }}
```

3.5.4 Deleting

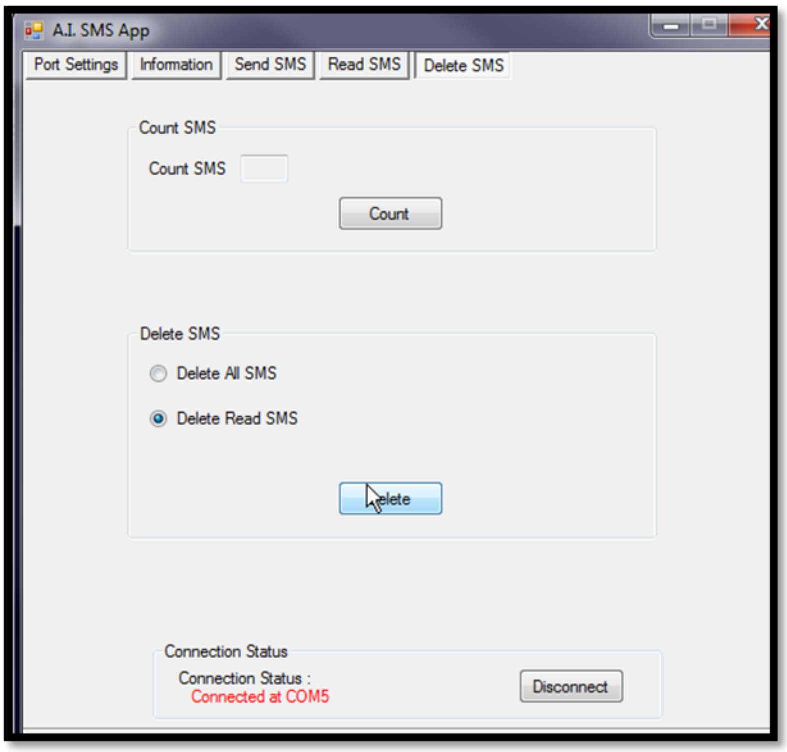


Fig 3.22: (AI SMS APP – Deleting)

```
  private void btnDeleteSMS_Click(object sender, EventArgs e)
        {
            try
            {
                //Count SMS
                int uCountSMS = objclsSMS.CountSMSmessages(this.port);
                if (uCountSMS > 0)
                {
DialogResult dr = MessageBox.Show("Are u sure u want to delete the SMS?", "Delete
confirmation", MessageBoxButtons.YesNo);

if (dr.ToString() == "Yes")
{
#region Delete SMS

if (this.rbDeleteAllSMS.Checked)
{
//...............................................Delete all SMS
.....................................................

#region Delete all SMS
string strCommand = "AT+CMGD=1,4";
if (objclsSMS.DeleteMsg(this.port, strCommand))
{
//MessageBox.Show("Messages has deleted successfuly ");
this.statusBar1.Text = "Messages has deleted successfuly";
}
else
{
//MessageBox.Show("Failed to delete messages ");
this.statusBar1.Text = "Failed to delete messages";
}
#endregion
}
else if (this.rbDeleteReadSMS.Checked)
{
//...........................................Delete Read SMS

#region Delete Read SMS
string strCommand = "AT+CMGD=1,3";
if (objclsSMS.DeleteMsg(this.port, strCommand))
{
//MessageBox.Show("Messages has deleted successfuly");
this.statusBar1.Text = "Messages has deleted successfuly";
}
else
{
//MessageBox.Show("Failed to delete messages ");
this.statusBar1.Text = "Failed to delete messages";
}
#endregion
}
#endregion
}
                }
            }
            catch (Exception ex)
            {
                ErrorLog(ex.Message);
            }}
```

3.5.5 SMS APP (complete source C# code)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.IO.Ports;

namespace SMSapplication
{
public partial class SMSapplication : Form
{

#region Constructor
public SMSapplication()
{
InitializeComponent();
}
#endregion

#region Private Variables
SerialPort port = new SerialPort();
clsSMS objclsSMS = new clsSMS();
ShortMessageCollection objShortMessageCollection = new
ShortMessageCollection();
#endregion

#region Private Methods

#region Write StatusBar
```

```
private void WriteStatusBar(string status)
{
try
{
statusBar1.Text = "Message: " + status;
}
catch (Exception ex)
{

}
}
#endregion

#endregion

#region Private Events

private void SMSapplication_Load(object sender, EventArgs e)
{
try
{
#region Display all available COM Ports
string[] ports = SerialPort.GetPortNames();

// Add all port names to the combo box:
foreach (string port in ports)
{
this.cboPortName.Items.Add(port);
}
#endregion

this.btnDisconnect.Enabled = false;
}
catch(Exception ex)
{
ErrorLog(ex.Message);
}
}
private void btnOK_Click(object sender, EventArgs e)
{
try
{
//Open communication port
this.port = objclsSMS.OpenPort(this.cboPortName.Text,
Convert.ToInt32(this.cboBaudRate.Text),
Convert.ToInt32(this.cboDataBits.Text),
Convert.ToInt32(this.txtReadTimeOut.Text),
Convert.ToInt32(this.txtWriteTimeOut.Text));
if (this.port != null)
{
this.gboPortSettings.Enabled = false;
//MessageBox.Show("Modem is connected at PORT " +
this.cboPortName.Text);
this.statusBar1.Text = "Modem is connected at PORT " +
this.cboPortName.Text;
this.lblConnectionStatus.Text = "Connected at " +
this.cboPortName.Text;
this.btnDisconnect.Enabled = true;
}

else
```

```
{
//MessageBox.Show("Invalid port settings");
this.statusBar1.Text = "Invalid port settings";
}
}
catch (Exception ex)
{
ErrorLog(ex.Message);
}
}
private void btnDisconnect_Click(object sender, EventArgs e)
{
try
{
this.gboPortSettings.Enabled = true;
objclsSMS.ClosePort(this.port);
this.lblConnectionStatus.Text = "Not Connected";
this.btnDisconnect.Enabled = false;
}
catch (Exception ex)
{
ErrorLog(ex.Message);
}
}
private void btnSendSMS_Click(object sender, EventArgs e)
{
//... Send SMS
try
{

if (objclsSMS.sendMsg(this.port, this.txtSIM.Text,
this.txtMessage.Text))
{
//MessageBox.Show("Message has sent successfully");
this.statusBar1.Text = "Message has sent successfully";
}
else
{
//MessageBox.Show("Failed to send message");
this.statusBar1.Text = "Failed to send message";
}

}
catch (Exception ex)
{
ErrorLog(ex.Message);
}
}
private void btnReadSMS_Click(object sender, EventArgs e)
{
lvwMessages.Items.Clear();
try
{
//count SMS
int uCountSMS = objclsSMS.CountSMSmessages(this.port);
if (uCountSMS > 0)
{

#region Command
string strCommand = "AT+CMGL=\"ALL\"";
```

```
if (this.rbReadAll.Checked)
{
strCommand = "AT+CMGL=\"ALL\"";
}
else if (this.rbReadUnRead.Checked)
{
strCommand = "AT+CMGL=\"REC UNREAD\"";
}
else if (this.rbReadStoreSent.Checked)
{
strCommand = "AT+CMGL=\"STO SENT\"";
}
else if (this.rbReadStoreUnSent.Checked)
{
strCommand = "AT+CMGL=\"STO UNSENT\"";
}
else if (this.ReadReadSMS.Checked)
{
strCommand = "AT+CMGL=\"REC READ\"";
}
#endregion

// If SMS exist then read SMS
#region Read SMS
//............................................ Read all SMS
..................................................
objShortMessageCollection = objclsSMS.ReadSMS(this.port, strCommand);
foreach (ShortMessage msg in objShortMessageCollection)
{

ListViewItem item = new ListViewItem(new string[] { msg.Index,
msg.Sent, msg.Sender, msg.Message });
item.Tag = msg;
lvwMessages.Items.Add(item);

}
#endregion

}
else
{
lvwMessages.Clear();
//MessageBox.Show("There is no message in SIM");
this.statusBar1.Text = "There is no message in SIM";

}
}
catch (Exception ex)
{
ErrorLog(ex.Message);
}
}
private void btnDeleteSMS_Click(object sender, EventArgs e)
{
try
{
//Count SMS
int uCountSMS = objclsSMS.CountSMSmessages(this.port);
if (uCountSMS > 0)
{
```

```
DialogResult dr = MessageBox.Show("Are u sure u want to delete the
SMS?", "Delete confirmation", MessageBoxButtons.YesNo);

if (dr.ToString() == "Yes")
{
#region Delete SMS

if (this.rbDeleteAllSMS.Checked)
{
//..........................................Delete all SMS
............................................

#region Delete all SMS
string strCommand = "AT+CMGD=1,4";
if (objclsSMS.DeleteMsg(this.port, strCommand))
{
//MessageBox.Show("Messages has deleted successfuly ");
this.statusBar1.Text = "Messages has deleted successfuly";
}
else
{
//MessageBox.Show("Failed to delete messages ");
this.statusBar1.Text = "Failed to delete messages";
}
#endregion

}
else if (this.rbDeleteReadSMS.Checked)
{
//..........................................Delete Read SMS
............................................

#region Delete Read SMS
string strCommand = "AT+CMGD=1,3";
if (objclsSMS.DeleteMsg(this.port, strCommand))
{
//MessageBox.Show("Messages has deleted successfuly");
this.statusBar1.Text = "Messages has deleted successfuly";
}
else
{
//MessageBox.Show("Failed to delete messages ");
this.statusBar1.Text = "Failed to delete messages";
}
#endregion

}

#endregion
}
}
}
catch (Exception ex)
{
ErrorLog(ex.Message);
}

}
private void btnCountSMS_Click(object sender, EventArgs e)
{
try
```

```
{
//Count SMS
int uCountSMS = objclsSMS.CountSMSmessages(this.port);
this.txtCountSMS.Text = uCountSMS.ToString();
}
catch (Exception ex)
{
ErrorLog(ex.Message);
}
}

#endregion

#region Error Log
public void ErrorLog(string Message)
{
StreamWriter sw = null;

try
{
WriteStatusBar(Message);

string sLogFormat = DateTime.Now.ToShortDateString().ToString() + " "
+ DateTime.Now.ToLongTimeString().ToString() + " ==> ";
//string sPathName = @"E:\";
string sPathName = @"SMSapplicationErrorLog_";

string sYear = DateTime.Now.Year.ToString();
string sMonth = DateTime.Now.Month.ToString();
string sDay = DateTime.Now.Day.ToString();

string sErrorTime = sDay + "-" + sMonth + "-" + sYear;

sw = new StreamWriter(sPathName + sErrorTime + ".txt", true);

sw.WriteLine(sLogFormat + Message);
sw.Flush();

}
catch (Exception ex)
{
//ErrorLog(ex.ToString());
}
finally
{
if (sw != null)
{
sw.Dispose();
sw.Close();
}
}

}
#endregion

private void tabSMSapplication_Selected(object sender,
TabControlEventArgs e)
{

string input = objclsSMS.ExecCommand(port, "ATI", 5000, "Failed to
execute command");
```

```
this.textBox1.Text = input;

input = objclsSMS.ExecCommand(port, "AT+CIMI", 5000, "Failed to
execute command");
this.textBox1.Text += "IMSI:" + input;

input = objclsSMS.ExecCommand(port, "AT+CPIN?", 5000, "Failed to
execute command");
this.textBox1.Text += "PIN:" + input;

input = objclsSMS.ExecCommand(port, "AT+CSCA?", 5000, "Failed to
execute command");
this.textBox1.Text += "SMC Adress:" + input;
}

private void tabInfo_Click(object sender, EventArgs e)
{

}

private void textBox1_TextChanged(object sender, EventArgs e)
{

}

private void button2_Click(object sender, EventArgs e)
{

string input = objclsSMS.ExecCommand(port, "ATI", 5000, "Failed to
execute command");
this.textBox1.Text = input;

input = objclsSMS.ExecCommand(port, "AT+CIMI", 5000, "Failed to
execute command");
this.textBox1.Text += "IMSI:" + input;

input = objclsSMS.ExecCommand(port, "AT+CPIN?", 5000, "Failed to
execute command");
this.textBox1.Text += "PIN:" + input;

input = objclsSMS.ExecCommand(port, "AT+CSCA?", 5000, "Failed to
execute command");
this.textBox1.Text += "SMC Adress:" + input;
}

private void button1_Click(object sender, EventArgs e)
{
string input = objclsSMS.ExecCommand(port,
"AT+CSCA=this.textBox3.Text", 5000, "Failed to execute command");


}


}
}

Class clsSMS.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.IO.Ports;
using System.Threading;
using System.Text.RegularExpressions;

namespace SMSapplication
{
public class clsSMS
{

#region Open and Close Ports
//Open Port
public SerialPort OpenPort(string p_strPortName, int p_uBaudRate, int
p_uDataBits, int p_uReadTimeout, int p_uWriteTimeout)
{
receiveNow = new AutoResetEvent(false);
SerialPort port = new SerialPort();

try
{
port.PortName = p_strPortName;                  //COM1
port.BaudRate = p_uBaudRate;                    //9600
port.DataBits = p_uDataBits;                    //8
port.StopBits = StopBits.One;                   //1
port.Parity = Parity.None; //None
port.ReadTimeout = p_uReadTimeout;              //300
port.WriteTimeout = p_uWriteTimeout;            //300
port.Encoding = Encoding.GetEncoding("iso-8859-1");
port.DataReceived += new
SerialDataReceivedEventHandler(port_DataReceived);
port.Open();
port.DtrEnable = true;
port.RtsEnable = true;
}
catch (Exception ex)
{
throw ex;
}
return port;
}

//Close Port
public void ClosePort(SerialPort port)
{
try
{
port.Close();
port.DataReceived -= new
SerialDataReceivedEventHandler(port_DataReceived);
port = null;
}
catch (Exception ex)
{
throw ex;
}
}

#endregion
```

```
//Execute AT Command
public string ExecCommand(SerialPort port,string command, int
responseTimeout, string errorMessage)
{
try
{

port.DiscardOutBuffer();
port.DiscardInBuffer();
receiveNow.Reset();
port.Write(command + "\r");

string input = ReadResponse(port, responseTimeout);
if ((input.Length == 0) || ((!input.EndsWith("\r\n> ")) &&
(!input.EndsWith("\r\nOK\r\n"))))
throw new ApplicationException("No success message was received.");
return input;
}
catch (Exception ex)
{
throw ex;
}
}

//Receive data from port
public void port_DataReceived(object sender,
SerialDataReceivedEventArgs e)
{
try
{
if (e.EventType == SerialData.Chars)
{
receiveNow.Set();
}
}
catch (Exception ex)
{
throw ex;
}
}
public string ReadResponse(SerialPort port,int timeout)
{
string buffer = string.Empty;
try
{
do
{
if (receiveNow.WaitOne(timeout, false))
{
string t = port.ReadExisting();
buffer += t;
}
else
{
if (buffer.Length > 0)
throw new ApplicationException("Response received is incomplete.");
else
throw new ApplicationException("No data received from phone.");
}
}
```

```
while (!buffer.EndsWith("\r\nOK\r\n") && !buffer.EndsWith("\r\n> ")
&& !buffer.EndsWith("\r\nERROR\r\n"));
}
catch (Exception ex)
{
throw ex;
}
return buffer;
}

#region Count SMS
public int CountSMSmessages(SerialPort port)
{
int CountTotalMessages = 0;
try
{

#region Execute Command

string recievedData = ExecCommand(port, "AT", 300, "No phone
connected at ");
recievedData = ExecCommand(port, "AT+CMGF=1", 300, "Failed to set
message format.");
String command = "AT+CPMS?";
recievedData = ExecCommand(port, command, 1000, "Failed to count SMS
message");
int uReceivedDataLength = recievedData.Length;

#endregion

#region If command is executed successfully
if ((recievedData.Length >= 5) && (recievedData.Contains("CPMS")))
{

#region Parsing SMS
string[] strSplit = recievedData.Split(',');
string strMessageStorageArea1 = strSplit[0];     //SM
string strMessageExist1 = strSplit[1];           //Msgs exist in SM
#endregion

#region Count Total Number of SMS In SIM
CountTotalMessages = Convert.ToInt32(strMessageExist1);
#endregion

}
#endregion

#region If command is not executed successfully
else if (recievedData.Contains("ERROR"))
{

#region Error in Counting total number of SMS
string recievedError = recievedData;
recievedError = recievedError.Trim();
recievedData = "Following error occured while counting the message" +
recievedError;
#endregion

}
#endregion
```

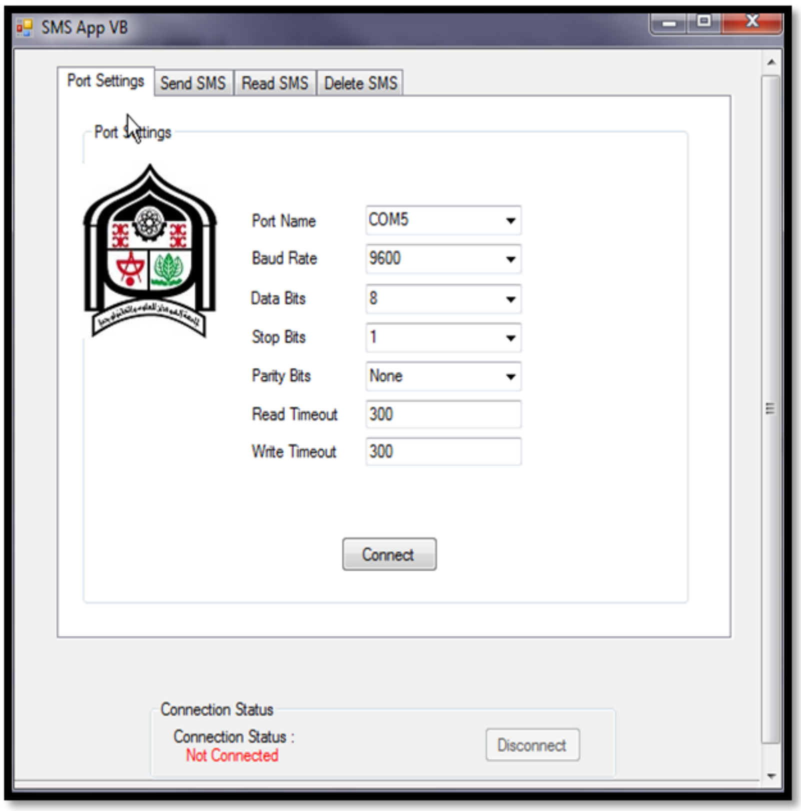
```
return CountTotalMessages;

}
catch (Exception ex)
{
throw ex;
}

}
#endregion

#region Read SMS

public AutoResetEvent receiveNow;

public ShortMessageCollection ReadSMS(SerialPort port, string
p_strCommand)
{

// Set up the phone and read the messages
ShortMessageCollection messages = null;
try
{

#region Execute Command
// Check connection
ExecCommand(port,"AT", 300, "No phone connected");
// Use message format "Text mode"
ExecCommand(port,"AT+CMGF=1", 300, "Failed to set message format.");
// Use character set "PCCP437"
//   ExecCommand(port,"AT+CSCS=\"PCCP437\"", 300, "Failed to set
character set.");
// Select SIM storage
ExecCommand(port,"AT+CPMS=\"SM\"", 300, "Failed to select message
storage.");
// Read the messages
string input = ExecCommand(port, p_strCommand, 5000, "Failed to read
the messages.");
#endregion

#region Parse messages
messages = ParseMessages(input);
#endregion

}
catch (Exception ex)
{
throw ex;
}

if (messages != null)
return messages;
else
return null;

}
public ShortMessageCollection ParseMessages(string input)
{
ShortMessageCollection messages = new ShortMessageCollection();
try
{
```

```
Regex r = new Regex(@"\+CMGL:
(\d+),""(.+)"",""(.+)"",(.*),""(.+)""\r\n(.+)\r\n");
Match m = r.Match(input);
while (m.Success)
{
ShortMessage msg = new ShortMessage();
//msg.Index = int.Parse(m.Groups[1].Value);
msg.Index = m.Groups[1].Value;
msg.Status = m.Groups[2].Value;
msg.Sender = m.Groups[3].Value;
msg.Alphabet = m.Groups[4].Value;
msg.Sent = m.Groups[5].Value;
msg.Message = m.Groups[6].Value;
messages.Add(msg);

m = m.NextMatch();
}

}
catch (Exception ex)
{
throw ex;
}
return messages;
}

#endregion

#region Send SMS

static AutoResetEvent readNow = new AutoResetEvent(false);

public bool sendMsg(SerialPort port, string PhoneNo, string Message)
{
bool isSend = false;

try
{

string recievedData = ExecCommand(port,"AT", 300, "No phone
connected");
recievedData = ExecCommand(port,"AT+CMGF=1", 300, "Failed to set
message format.");
String command = "AT+CMGS=\"" + PhoneNo + "\"";
recievedData = ExecCommand(port,command, 300, "Failed to accept
phoneNo");
command = Message + char.ConvertFromUtf32(26) + "\r";
recievedData = ExecCommand(port,command, 3000, "Failed to send
message"); //3 seconds
if (recievedData.EndsWith("\r\nOK\r\n"))
{
isSend = true;
}
else if (recievedData.Contains("ERROR"))
{
isSend = false;
}
return isSend;
}
catch (Exception ex)
{
```

```
throw ex;
}

}
static void DataReceived(object sender, SerialDataReceivedEventArgs
e)
{
try
{
if (e.EventType == SerialData.Chars)
readNow.Set();
}
catch (Exception ex)
{
throw ex;
}
}

#endregion

#region Delete SMS
public bool DeleteMsg(SerialPort port , string p_strCommand)
{
bool isDeleted = false;
try
{

#region Execute Command
string recievedData = ExecCommand(port,"AT", 300, "No phone
connected");
recievedData = ExecCommand(port,"AT+CMGF=1", 300, "Failed to set
message format.");
String command = p_strCommand;
recievedData = ExecCommand(port,command, 300, "Failed to delete
message");
#endregion

if (recievedData.EndsWith("\r\nOK\r\n"))
{
isDeleted = true;
}
if (recievedData.Contains("ERROR"))
{
isDeleted = false;
}
return isDeleted;
}
catch (Exception ex)
{
throw ex;
}

}
#endregion

}
}
Class shortMessages.cs
using System;
using System.Collections.Generic;
using System.Text;
```

```
namespace SMSapplication
{
public class ShortMessage
{

#region Private Variables
private string index;
private string status;
private string sender;
private string alphabet;
private string sent;
private string message;
#endregion

#region Public Properties
public string Index
{
get { return index;}
set { index = value;}
}
public string Status
{
get { return status;}
set { status = value;}
}
public string Sender
{
get { return sender;}
set { sender = value;}
}
public string Alphabet
{
get { return alphabet;}
set { alphabet = value;}
}
public string Sent
{
get { return sent;}
set { sent = value;}
}
public string Message
{
get { return message;}
set { message = value;}
}
#endregion

}

public class ShortMessageCollection : List<ShortMessage>
{
}
}
```

3.6 Case#4: Managing SMS via Visual Basic



Port Name: COM5
Baud Rate: 9600
Data Bits: 8
Stop Bits: 1
Parity Bits: None
Read Timeout: 300
Write Timeout: 300

Connection Status :
Not Connected

Fig 3.23: (AI SMS APP – VB)

3.6.1 Connecting to serial port

```
        Public Function OpenPort(ByVal strPortName As String, ByVal strBaudRate
As String) As SerialPort
            receiveNow = New AutoResetEvent(False)
            Dim port As New SerialPort()
            port.PortName = strPortName
            port.BaudRate = Convert.ToInt32(strBaudRate)
            'A.I
            port.DataBits = 8
            port.StopBits = StopBits.One
            port.Parity = Parity.None
            port.ReadTimeout = 300
            port.WriteTimeout = 300
            port.Encoding = Encoding.GetEncoding("iso-8859-1")
            AddHandler port.DataReceived, New
SerialDataReceivedEventHandler(AddressOf port_DataReceived)
            port.Open()
            port.DtrEnable = True
            port.RtsEnable = True
            Return port
        End Function
```

3.6.2 Sending

```
#Region "Send SMS"

        Shared readNow As New AutoResetEvent(False)
        Public Function sendMsg(ByVal port As SerialPort, ByVal
strPortName As String, ByVal strBaudRate As String, ByVal PhoneNo As
String, ByVal Message As String) As Boolean
            Dim isSend As Boolean = False
            Try

                'this.port = OpenPort(strPortName,strBaudRate);
                Dim recievedData As String = ExecCommand(port, "AT",
300, "No phone connected at " & strPortName & ".")
                recievedData = ExecCommand(port, "AT+CMGF=1", 300,
"Failed to set message format.")
                Dim command As [String] = "AT+CMGS=""" & PhoneNo &
""""
                recievedData = ExecCommand(port, command, 300,
"Failed to accept phoneNo")
                command = Message & Char.ConvertFromUtf32(26) & vbCr
                recievedData = ExecCommand(port, command, 3000,
"Failed to send message")
                '3 seconds
                If recievedData.EndsWith(vbCr & vbLf & "OK" & vbCr &
vbLf) Then
                    recievedData = "Message sent successfully"
                    isSend = True
                ElseIf recievedData.Contains("ERROR") Then
                    Dim recievedError As String = recievedData
                    recievedError = recievedError.Trim()
                    recievedData = "Following error occured while
sending the message" & recievedError
                    isSend = False
                End If
                Return isSend
            Catch ex As Exception
                Throw New Exception(ex.Message)
            Finally
                'port.Close();
                'port.DataReceived -= new
SerialDataReceivedEventHandler(port_DataReceived);
                'port = null;
                If port IsNot Nothing Then
                End If
            End Try
        End Function
```

3.6.3 Reading

```
  Public Function ReadSMS(port As SerialPort, p_strCommand As String)
As ShortMessageCollection
            ' Set up the phone and read the messages
            Dim messages As ShortMessageCollection = Nothing
            Try
                ExecCommand(port, "AT", 300, "No phone connected")
                ' Use message format "Text mode"
ExecCommand(port, "AT+CMGF=1", 300, "Failed to set message format.")
                                ' Select SIM storage
ExecCommand(port, "AT+CPMS=""SM""", 300, "Failed to select message
storage.")
                ' Read the messages
Dim input As String = ExecCommand(port, p_strCommand, 5000, "Failed
to read the messages.")
```

```
                '#End Region
                '#Region "Parse messages"
                '#End Region
                messages = ParseMessages(input)
            Catch ex As Exception
                Throw ex
            End Try
            If messages IsNot Nothing Then
                Return messages
            Else
                Return Nothing
            End If
        End
Function
```

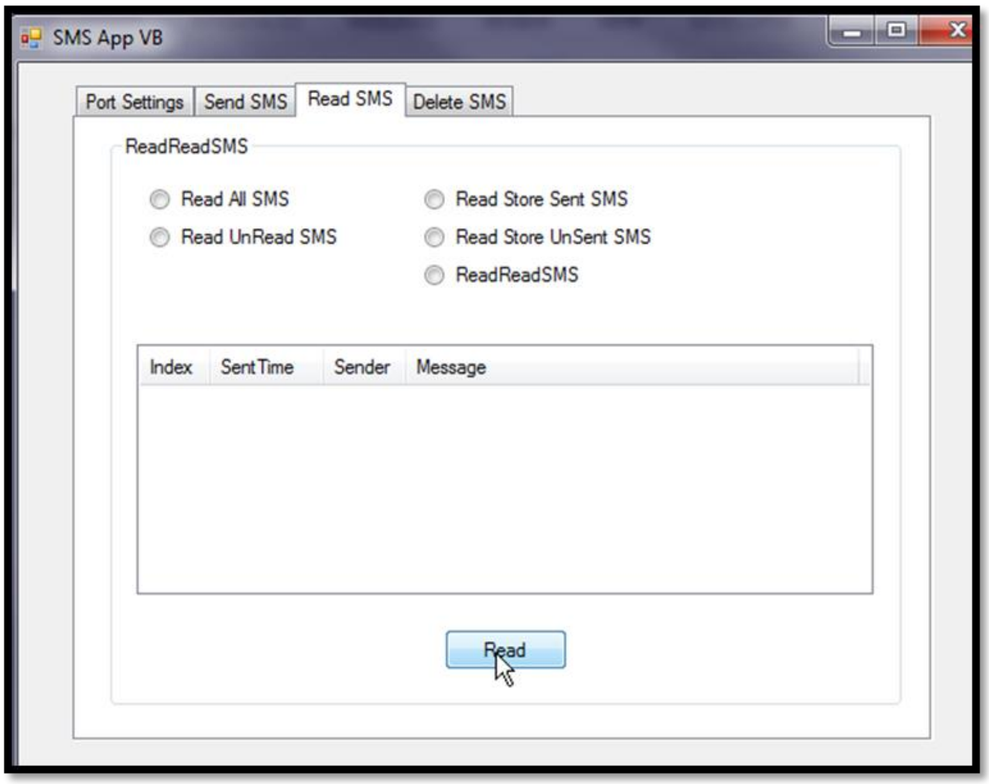


Fig 3.24: (AI SMS APP – VB Reading Tab)

```
    Private Sub btnReadSMS_Click(sender As Object, e As EventArgs)
Handles btnReadSMS.Click
        lvwMessages.Items.Clear()
        Try
            'count SMS
            Dim uCountSMS As Integer
            uCountSMS = objclsSMS.CountSMSmessages(Me.port)

            If uCountSMS > 0 Then

                '#Region "Command"
                Dim strCommand As String = "AT+CMGL=""ALL"""

                If Me.rbReadAll.Checked Then
                    strCommand = "AT+CMGL=""ALL"""
                ElseIf Me.rbReadUnRead.Checked Then
                    strCommand = "AT+CMGL=""REC UNREAD"""
                ElseIf Me.rbReadStoreSent.Checked Then
                    strCommand = "AT+CMGL=""STO SENT"""
                ElseIf Me.rbReadStoreUnSent.Checked Then
                    strCommand = "AT+CMGL=""STO UNSENT"""
                ElseIf Me.ReadReadSMS.Checked Then
                    strCommand = "AT+CMGL=""REC READ"""
                End If
                '#End Region
```

3.6.4 Deleting

```
#Region "Delete SMS"
        Public Function DeleteMsg(port As SerialPort, p_strCommand As
String) As Boolean
            Dim isDeleted As Boolean = False
            Try
                Dim recievedData As String = ExecCommand(port, "AT",
300, "No phone connected")
                recievedData = ExecCommand(port, "AT+CMGF=1", 300,
"Failed to set message format.")
                Dim command As [String] = p_strCommand
                recievedData = ExecCommand(port, command, 300,
"Failed to delete message")
                '#End Region
                If recievedData.EndsWith(vbCr & vbLf & "OK" & vbCr &
vbLf) Then
                    isDeleted = True
                End If
                If recievedData.Contains("ERROR") Then
                    isDeleted = False
                End If
                Return isDeleted
            Catch ex As Exception
                Throw ex
            End Try
        End Function
#End Region
```

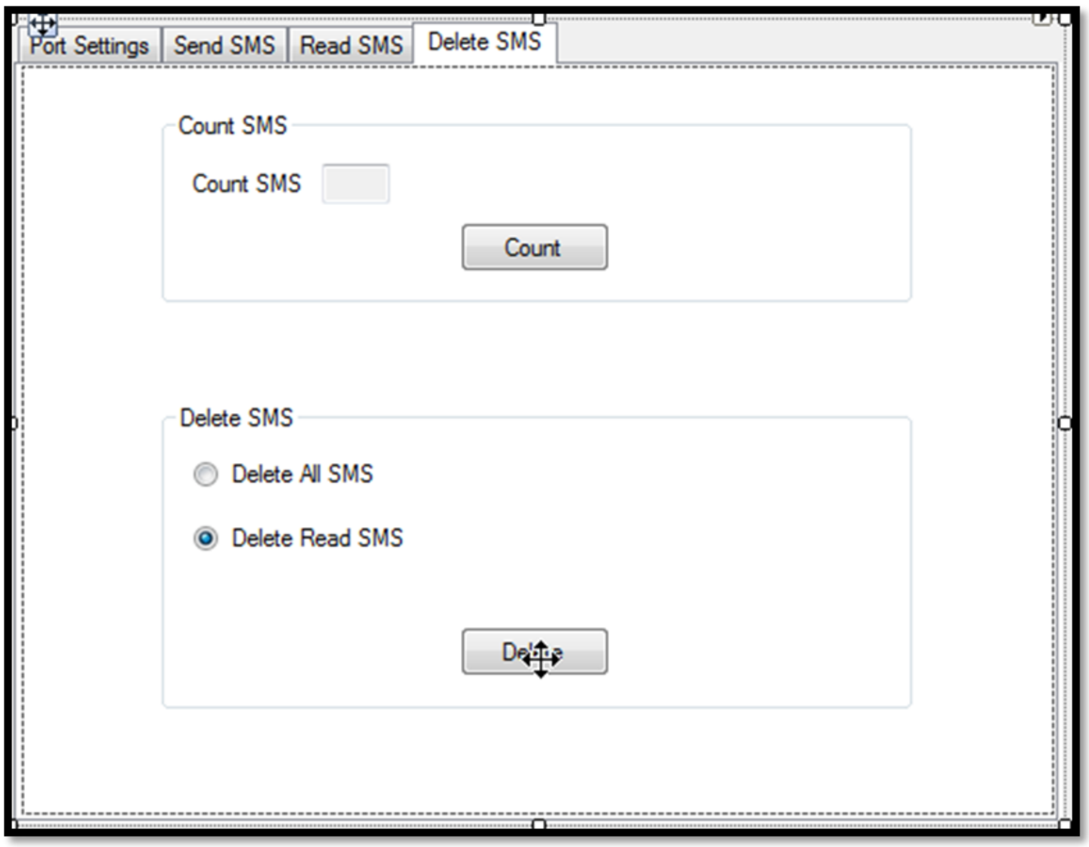


Fig 3.25: (AI SMS APP – VB Deleting Tab)

```
Private Sub btnDeleteSMS_Click(sender As Object, e As EventArgs)
Handles btnDeleteSMS.Click
        Try
            'Count SMS
            Dim uCountSMS As Integer =
objclsSMS.CountSMSmessages(Me.port)
            If uCountSMS > 0 Then
                Dim dr As DialogResult = MessageBox.Show("Are u sure
u want to delete the SMS?", "Delete confirmation",
MessageBoxButtons.YesNo)

                If dr.ToString() = "Yes" Then
                    '#Region "Delete SMS"

                    If Me.rbDeleteAllSMS.Checked Then
                        'Delete all SMS

                        '#Region "Delete all SMS"
                        Dim strCommand As String = "AT+CMGD=1,4"
                        If objclsSMS.DeleteMsg(Me.port, strCommand)
Then
'MessageBox.Show("Messages has deleted successfuly ");
Me.statusBar1.Text = "Messages has deleted successfuly"
                        Else
                            'MessageBox.Show("Failed to delete
messages ");
                            Me.statusBar1.Text = "Failed to delete
messages"
                            '#End Region
                        End If
                    ElseIf Me.rbDeleteReadSMS.Checked Then

'................................................Delete Read SMS
                        '#Region "Delete Read SMS"
                        Dim strCommand As String = "AT+CMGD=1,3"
                        If objclsSMS.DeleteMsg(Me.port, strCommand)
Then
'MessageBox.Show("Messages has deleted successfuly");
Me.statusBar1.Text = "Messages has deleted successfuly"
                        Else
                            'MessageBox.Show("Failed to delete
messages ");
                            Me.statusBar1.Text = "Failed to delete
messages"
                            '#End Region

                        End If

                        '#End Region
                    End If
                End If
            End If
        Catch ex As Exception
            ErrorLog(ex.Message)
        End Try

    End Sub
```

3.6.5 SMS APP (complete source VB code)

```
I. Form1(MAIN)

Imports System.IO.Ports
Imports System.IO
Public Class tabSMSapplication
    Dim port As New SerialPort()
    Dim objclsSMS As New MY_SMS_Application.clsSMS()
    Dim objShortMessageCollection As New
MY_SMS_Application.ShortMessageCollection()
    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
    End Sub
    Private Sub gboConnectionStatus_Enter(ByVal sender As
System.Object, ByVal e As System.EventArgs)
    End Sub
    Private Sub WriteStatusBar(ByVal status As String)
        Try
            statusBar1.Text = "Message: " & status

        Catch ex As Exception
        End Try
    End Sub

    Public Sub ErrorLog(ByVal Message As String)
        Dim sw As StreamWriter = Nothing
        Try
            WriteStatusBar(Message)
            Dim sLogFormat As String =
DateTime.Now.ToShortDateString().ToString() & " " &
DateTime.Now.ToLongTimeString().ToString() & " ==> "
```

```
            'string sPathName = @"E:\";
            Dim sPathName As String = "SMSapplicationErrorLog_"
            Dim sYear As String = DateTime.Now.Year.ToString()
            Dim sMonth As String = DateTime.Now.Month.ToString()
            Dim sDay As String = DateTime.Now.Day.ToString()
            Dim sErrorTime As String = sDay & "-" & sMonth & "-" &
sYear
            sw = New StreamWriter(sPathName & sErrorTime & ".txt",
True)
            sw.WriteLine(sLogFormat & Message)
            sw.Flush()
            'ErrorLog(ex.ToString());
        Catch ex As Exception
        Finally
            If sw IsNot Nothing Then
                sw.Dispose()
                sw.Close()
            End If
        End Try
    End Sub
    Private Sub btnOK_Click_1(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnOK.Click
        Try
            'Open communication port
            Me.port = objclsSMS.OpenPort(Me.cboPortName.Text,
Convert.ToInt32(Me.cboBaudRate.Text))
            If Me.port IsNot Nothing Then
                Me.gboPortSettings.Enabled = False
                'MessageBox.Show("Modem is connected at PORT " +
this.cboPortName.Text);
                Me.statusBar1.Text = "Modem is connected at PORT " &
Convert.ToString(Me.cboPortName.Text)
                'Add tab pages
                '  Me.TabControl1.TabPages.Add(tbSendSMS)
                '    Me.TabControl1.TabPages.Add(tbReadSMS)
                '    Me.TabControl1.TabPages.Add(tbDeleteSMS)
                Me.lblConnectionStatus.Text = "Connected at " &
Convert.ToString(Me.cboPortName.Text)
                Me.btnDisconnect.Enabled = True
            Else
                'MessageBox.Show("Invalid port settings");
                Me.statusBar1.Text = "Invalid port settings"
            End If
        Catch ex As Exception
            ErrorLog(ex.Message)
        End Try
    End Sub
    Private Sub btnSendSMS_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnSendSMS.Click

        ' Send SMS.....
        Try

            If objclsSMS.sendMsg(Me.port,
Me.cboPortName.Text,cboBaudRate.Text,Me.txtSIM.Text,
Me.txtMessage.Text) Then
                'MessageBox.Show("Message has sent successfully");
                Me.statusBar1.Text = "Message has sent successfully"
            Else
                'MessageBox.Show("Failed to send message");
                Me.statusBar1.Text = "Failed to send message"
            End If
        Catch ex As Exception
            ErrorLog(ex.Message)
        End Try
    End Sub
    Private Sub btnReadSMS_Click(sender As Object, e As EventArgs)
Handles btnReadSMS.Click
        lvwMessages.Items.Clear()
        Try
            'count SMS
            Dim uCountSMS As Integer
            uCountSMS = objclsSMS.CountSMSmessages(Me.port)

            If uCountSMS > 0 Then

                '#Region "Command"
                Dim strCommand As String = "AT+CMGL=""ALL"""

                If Me.rbReadAll.Checked Then
                    strCommand = "AT+CMGL=""ALL"""
                ElseIf Me.rbReadUnRead.Checked Then
                    strCommand = "AT+CMGL=""REC UNREAD"""
                ElseIf Me.rbReadStoreSent.Checked Then
                    strCommand = "AT+CMGL=""STO SENT"""
                ElseIf Me.rbReadStoreUnSent.Checked Then
                    strCommand = "AT+CMGL=""STO UNSENT"""
                ElseIf Me.ReadReadSMS.Checked Then
                    strCommand = "AT+CMGL=""REC READ"""
                End If
                '#End Region

                ' If SMS exist then read SMS
                '#Region "Read SMS"
                ' Read all SMS
 objShortMessageCollection = objclsSMS.ReadSMS(Me.port, strCommand)
 For Each msg As MY_SMS_Application.ShortMessage In
objShortMessageCollection
 Dim item As New ListViewItem(New String() {msg.Index, msg.Sent,
msg.Sender, msg.Message})
                    item.Tag = msg
                    lvwMessages.Items.Add(item)
                    '#End Region
                Next
            Else
                lvwMessages.Clear()
                'MessageBox.Show("There is no message in SIM");

                Me.statusBar1.Text = "There is no message in SIM"
            End If
        Catch ex As Exception
            ErrorLog(ex.Message)
        End Try
    End Sub
    Private Sub btnCountSMS_Click(sender As Object, e As EventArgs)
Handles btnCountSMS.Click
        Try
            'Count SMS
            Dim uCountSMS As Integer =
objclsSMS.CountSMSmessages(Me.port)
            Me.txtCountSMS.Text = uCountSMS.ToString()
        Catch ex As Exception
            ErrorLog(ex.Message)
        End Try
    End Sub
Private Sub btnDeleteSMS_Click(sender As Object, e As EventArgs)
Handles btnDeleteSMS.Click
        Try
            'Count SMS
            Dim uCountSMS As Integer =
objclsSMS.CountSMSmessages(Me.port)
            If uCountSMS > 0 Then
                Dim dr As DialogResult = MessageBox.Show("Are u sure
u want to delete the SMS?", "Delete confirmation",
MessageBoxButtons.YesNo)
                If dr.ToString() = "Yes" Then
                    '#Region "Delete SMS"
                    If Me.rbDeleteAllSMS.Checked Then
                        'Delete all SMS

                        '#Region "Delete all SMS"
                        Dim strCommand As String = "AT+CMGD=1,4"
                        If objclsSMS.DeleteMsg(Me.port, strCommand)
Then
                            'MessageBox.Show("Messages has deleted
successfuly ");
                            Me.statusBar1.Text = "Messages has
deleted successfuly"
                        Else
                            'MessageBox.Show("Failed to delete
messages ");
                            Me.statusBar1.Text = "Failed to delete
messages"
                            '#End Region
                        End If
                    ElseIf Me.rbDeleteReadSMS.Checked Then
                        'Delete Read SMS
                        '#Region "Delete Read SMS"
                        Dim strCommand As String = "AT+CMGD=1,3"
                        If objclsSMS.DeleteMsg(Me.port, strCommand)
Then
'MessageBox.Show("Messages has deleted successfuly");
Me.statusBar1.Text = "Messages has deleted successfuly"
                        Else
                            'MessageBox.Show("Failed to delete
messages ");
                            Me.statusBar1.Text = "Failed to delete
messages"
                            '#End Region
                        End If
                        '#End Region
                    End If
                End If
            End If
        Catch ex As Exception
            ErrorLog(ex.Message)
        End Try

    End Sub
```

```
Private Sub btnDisconnect_Click(sender As Object, e As EventArgs)
Handles btnDisconnect.Click
        Try
            Me.gboPortSettings.Enabled = True
            objclsSMS.ClosePort(Me.port)
            Me.lblConnectionStatus.Text = "Not Connected"
            Me.btnDisconnect.Enabled = False
        Catch ex As Exception
            ErrorLog(ex.Message)
        End Try
    End Sub
End Class
```

I. clsSMS

```
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.IO.Ports
Imports System.Threading
Imports System.Text.RegularExpressions
Namespace MY_SMS_Application
    Public Class clsSMS
        'public SerialPort port;

        'Open Port
        Public Function OpenPort(ByVal strPortName As String, ByVal
strBaudRate As String) As SerialPort
            receiveNow = New AutoResetEvent(False)
            Dim port As New SerialPort()
            port.PortName = strPortName
            port.BaudRate = Convert.ToInt32(strBaudRate)
            'A.I
            port.DataBits = 8
            port.StopBits = StopBits.One
            port.Parity = Parity.None
            port.ReadTimeout = 300
            port.WriteTimeout = 300
            port.Encoding = Encoding.GetEncoding("iso-8859-1")
            AddHandler port.DataReceived, New
SerialDataReceivedEventHandler(AddressOf port_DataReceived)
            port.Open()
            port.DtrEnable = True
            port.RtsEnable = True
            Return port
        End Function

        'Close Port
        Public Sub ClosePort(ByVal port As SerialPort)
            port.Close()
            RemoveHandler port.DataReceived, New
SerialDataReceivedEventHandler(AddressOf port_DataReceived)
            port = Nothing
        End Sub

        'Execute AT Command
        Public Function ExecCommand(ByVal port As SerialPort, ByVal
command As String, ByVal responseTimeout As Integer, ByVal
errorMessage As String) As String
```

```
            Try
                '  receiveNow = New AutoResetEvent(True)
                port.DiscardOutBuffer()
                port.DiscardInBuffer()
                receiveNow.Reset()
                port.Write(command + vbCr)

                'Thread.Sleep(3000)
                Dim input As String = ReadResponse(port, 
responseTimeout)
                If (input.Length = 0) OrElse ((Not 
input.EndsWith(vbCr & vbLf & "> ")) AndAlso (Not input.EndsWith(vbCr 
& vbLf & "OK" & vbCr & vbLf))) Then
                    Throw New ApplicationException("No success 
message was received.")
                End If

                Return input
            Catch ex As Exception
                Throw New ApplicationException(errorMessage, ex)
            End Try
        End Function

        'Receive data from port
        Public Sub port_DataReceived(ByVal sender As Object, ByVal e 
As SerialDataReceivedEventArgs)
            If e.EventType = SerialData.Chars Then
                receiveNow.[Set]()
            End If
        End Sub
        Public Function ReadResponse(ByVal port As SerialPort, ByVal 
timeout As Integer) As String
            Dim buffer As String = String.Empty
            Do
                If receiveNow.WaitOne(timeout, False) Then
                    Dim t As String = port.ReadExisting()
                    buffer += t
                Else
                    If buffer.Length > 0 Then
                        Throw New ApplicationException("Response 
received is incomplete.")
                    Else
                        Throw New ApplicationException("No data 
received from phone.")
                    End If
                End If
            Loop While Not buffer.EndsWith(vbCr & vbLf & "OK" & vbCr 
& vbLf) AndAlso Not buffer.EndsWith(vbCr & vbLf & "> ") AndAlso Not 
buffer.EndsWith(vbCr & vbLf & "ERROR" & vbCr & vbLf)
            Return buffer
        End Function
#Region "Count SMS"
        Public Function CountSMSmessages(port As SerialPort) As 
Integer
            Dim CountTotalMessages As Integer = 0
            Try

                '#Region "Execute Command"

                Dim recievedData As String = ExecCommand(port, "AT", 
300, "No phone connected at ")
```

```
                recievedData = ExecCommand(port, "AT+CMGF=1", 300,
"Failed to set message format.")
                Dim command As [String] = "AT+CPMS?"
                recievedData = ExecCommand(port, command, 1000,
"Failed to count SMS message")
                Dim uReceivedDataLength As Integer =
recievedData.Length

                '#End Region

                '#Region "If command is executed successfully"
                If (recievedData.Length >= 5) AndAlso
(recievedData.Contains("CPMS")) Then

                    '#Region "Parsing SMS"
                    Dim strSplit As String() =
recievedData.Split(","c)
                    Dim strMessageStorageArea1 As String =
strSplit(0)
                    'SM
                    Dim strMessageExist1 As String = strSplit(1)
                    'Msgs exist in SM
                    '#End Region
                    '#Region "Count Total Number of SMS In SIM"
                    '#End Region

                    CountTotalMessages =
Convert.ToInt32(strMessageExist1)
                    '#End Region

                    '#Region "If command is not executed
successfully"
                ElseIf recievedData.Contains("ERROR") Then

                    '#Region "Error in Counting total number of SMS"
                    Dim recievedError As String = recievedData
                    recievedError = recievedError.Trim()
                    '#End Region

                    recievedData = "Following error occured while
counting the message" & recievedError
                End If
                '#End Region


                Return CountTotalMessages
            Catch ex As Exception
                Throw ex
            End Try

        End Function
#End Region

#Region "Read SMS"

        Public receiveNow As AutoResetEvent

        Public Function ReadSMS(port As SerialPort, p_strCommand As
String) As ShortMessageCollection

            ' Set up the phone and read the messages
            Dim messages As ShortMessageCollection = Nothing
            Try

                '#Region "Execute Command"
                ' Check connection
                ExecCommand(port, "AT", 300, "No phone connected")
                ' Use message format "Text mode"
                ExecCommand(port, "AT+CMGF=1", 300, "Failed to set
message format.")
                ' Use character set "PCCP437"
                '   ExecCommand(port,"AT+CSCS=\"PCCP437\"", 300,
"Failed to set character set.");
                ' Select SIM storage
                ExecCommand(port, "AT+CPMS=""SM""", 300, "Failed to
select message storage.")
                ' Read the messages
                Dim input As String = ExecCommand(port, p_strCommand,
5000, "Failed to read the messages.")
                '#End Region

                '#Region "Parse messages"
                '#End Region

                messages = ParseMessages(input)
            Catch ex As Exception
                Throw ex
            End Try

            If messages IsNot Nothing Then
                Return messages
            Else
                Return Nothing
            End If

        End Function
        Public Function ParseMessages(input As String) As
ShortMessageCollection
            Dim messages As New ShortMessageCollection()
            Try
                Dim r As New Regex("\+CMGL:
(\d+),""(.+)"",""(.+)"",(.*),""(.+)""\r\n(.+)\r\n")
                Dim m As Match = r.Match(input)
                While m.Success
                    Dim msg As New ShortMessage()
                    'msg.Index = int.Parse(m.Groups[1].Value);
                    msg.Index = m.Groups(1).Value
                    msg.Status = m.Groups(2).Value
                    msg.Sender = m.Groups(3).Value
                    msg.Alphabet = m.Groups(4).Value
                    msg.Sent = m.Groups(5).Value
                    msg.Message = m.Groups(6).Value
                    messages.Add(msg)

                    m = m.NextMatch()

                End While
            Catch ex As Exception
                Throw ex
            End Try
            Return messages
        End Function
```

```
#End Region

#Region "Send SMS"

        Shared readNow As New AutoResetEvent(False)

        Public Function sendMsg(ByVal port As SerialPort, ByVal
strPortName As String, ByVal strBaudRate As String, ByVal PhoneNo As
String, ByVal Message As String) As Boolean
            Dim isSend As Boolean = False
            Try

                'this.port = OpenPort(strPortName,strBaudRate);
                Dim recievedData As String = ExecCommand(port, "AT",
300, "No phone connected at " & strPortName & ".")
                recievedData = ExecCommand(port, "AT+CMGF=1", 300,
"Failed to set message format.")
                Dim command As [String] = "AT+CMGS=""" & PhoneNo &
""""
                recievedData = ExecCommand(port, command, 300,
"Failed to accept phoneNo")
                command = Message & Char.ConvertFromUtf32(26) & vbCr
                recievedData = ExecCommand(port, command, 3000,
"Failed to send message")
                '3 seconds
                If recievedData.EndsWith(vbCr & vbLf & "OK" & vbCr &
vbLf) Then
                    recievedData = "Message sent successfully"
                    isSend = True
                ElseIf recievedData.Contains("ERROR") Then
                    Dim recievedError As String = recievedData
                    recievedError = recievedError.Trim()
                    recievedData = "Following error occured while
sending the message" & recievedError
                    isSend = False
                End If
                Return isSend
            Catch ex As Exception
                Throw New Exception(ex.Message)
            Finally
                'port.Close();
                'port.DataReceived -= new
SerialDataReceivedEventHandler(port_DataReceived);
                'port = null;
                If port IsNot Nothing Then
                End If
            End Try
        End Function
        Private Shared Sub DataReceived(ByVal sender As Object, ByVal
e As SerialDataReceivedEventArgs)
            If e.EventType = SerialData.Chars Then
                readNow.[Set]()
            End If
        End Sub

#End Region

#Region "Delete SMS"
        Public Function DeleteMsg(port As SerialPort, p_strCommand As
String) As Boolean
```

```
            Dim isDeleted As Boolean = False
            Try

                '#Region "Execute Command"
                Dim recievedData As String = ExecCommand(port, "AT",
300, "No phone connected")
                recievedData = ExecCommand(port, "AT+CMGF=1", 300,
"Failed to set message format.")
                Dim command As [String] = p_strCommand
                recievedData = ExecCommand(port, command, 300,
"Failed to delete message")
                '#End Region

                If recievedData.EndsWith(vbCr & vbLf & "OK" & vbCr &
vbLf) Then
                    isDeleted = True
                End If
                If recievedData.Contains("ERROR") Then
                    isDeleted = False
                End If
                Return isDeleted
            Catch ex As Exception
                Throw ex
            End Try

        End Function
#End Region

    End Class
End Namespace
```

4. Results & Discussions

4.1 Results

We built a Three programs using Three different Programing Languages, (MATLAB, C#, VB) each program built to manage GSM Modem for managing SMS (sending, receiving, listing Messages, deleting) all this via AT commands.

Basic instructions of each program shown in the Table (4.1) below.

Table 4.1: Basic instruction for managing SMS via AT commands

Command	MATLAB	C#	VB
Changing to text mode	fprintf(s,'%s', 'AT+CMGF=1');	port.Write('AT+CMGF=1' + "\r");	port.Write('AT+CMGF=1' & vbCr)
	fprintf(s,'%s', char(13));		
Sending	fprintf(s,'%s', 'AT+CMGS="0912xxxxxx"');	port.Write('AT+CMGS="0912xxxxxx"' + "\r");	port.Write(AT+CMGS=" & PhoneNo & """" & vbCr)
	fprintf(s,'%s', char(13));	port.Write('This is a test msg''+ char(26) + "\r");	port.Write('This is a test msg' & Char.ConvertFromUtf32(26) & vbCr)
	fprintf(s,'%s', 'This is a test msg');		
	fprintf(s,'%s', char(26));		
Listing	fprintf(s,'%s', 'AT+CMGF=1');	port.Write('AT+CMGF=1' + "\r");	port.Write('AT+CMGF=1' & vbCr)
	fprintf(s,'%s', char(13));		
	fprintf(s,'%s', 'at+cmgl="all"');	port.Write('at+cmgl="all"' + "\r");	port.Write('AT+CMGL="all"' & vbCr)
	fprintf(s,'%s', char(13));		
Reading	fprintf(s,'%s', 'AT+CMGF=1');	port.Write('AT+CMGF=1' + "\r");	port.Write('AT+CMGF=1' & vbCr)

	fprintf(s,'%s', char(13));		
	fprintf(s,'%s', 'at+cmgr=4');	port.Write('AT+CMGR=4' + "\r");	port.Write('AT+CMGR=4' & vbCr)
	fprintf(s,'%s', char(13));		
Reading Results	BytesAvail=s.BytesAvailable;	string t = port.ReadExisting();	Dim t As String = port.ReadExisting()
	A=fread(s,BytesAvail,'char'); end	this.textBox1.Text += t;	me.textBox1.Text += t;
	A;		
	sprintf('%c', A)		
Serial Port	s = serial('COM5');	SerialPort port = new SerialPort();	Dim port As New SerialPort()
	s.baudrate=9600;	port.PortName = p_strPortName; //COM1	port.PortName = strPortName
	fopen(s);	port.BaudRate = p_uBaudRate; //9600	port.BaudRate = Convert.ToInt32(strBaudRate)
	fclose(s)	port.DataBits = p_uDataBits; //8	port.DataBits = 8

4.2 Discussion

The objective of this project is Building a Table of comparison, for the different applications that used the AT commands.

As a result we successfully built three programs with different capabilities.

All three programs could be used to send AT commands to any GSM modem, for managing Short Message Service, however from testing results MATLAB shown some draw backs, sometimes it doesn’t give the right results every time, on the other hand the higher languages (C#, VB) programs tend to provide more flexible and user friendly interface, besides providing independent executable files, that could be run from the OS directly without need for the creation environment not like MATLAB.

In the operation level they all using the same AT commands for doing similar tasks.

This application is applying AT commands to Send SMS complying with GSM standards, which allows it to be used for many purposes starting from remote controlling, text chatting, up to broadcasting news.

It is also could be used as standard to build over it many other AT commands for Voice, FAX etc...)

5. Conclusion & Recommendations

5.1 Conclusion

We built a Three programs using Three different Programing Languages, (MATLAB, C#, VB) each program built to manage GSM Modem for managing SMS (sending, receiving, listing Messages, deleting) all this via AT commands.

This application is applying AT commands to Send SMS complying with GSM standards, which allows it to be used for many purposes starting from remote controlling, text chatting, up to broadcasting news.

5.2 Recommendations

AT commands is one of the very interesting topics, which reflecting the interconnecting of the GSM world with the computers.

A lot of research and application could be done here, which will lead to accurate controlling if used for remote controlling for instance, and could be used to build independent interface for USB modem, so you don't have to use the one which you had been given by the Service provider.

And finally it opens new horizons

Giving the PC ability to manage SMS, i.e. (send, receive, etc...) Will open a whole new opportunities of creating new services, commercial, bulk SMS, alarm systems … etc.