REMOTE CONTROL ON GSM SYSTEM

M.S. THESIS
Control and Computer Eng. Bülent ÖNEN

Department : CONTROL AND COMPUTER ENGINEERING
Program : CONTROL AND COMPUTER ENGINEERING

TC. VÜKSEKÖĞRETİM BÜRLÜJ
DOKÜMantasyon Merkezi

FEBRUARY 1999
GSM SİSTEMLERİNDE UZAKTAN KONTROL

YÜKSEK LİSANS TEZİ
Kontrol ve Bilgisayar Müh. Bülent ÖNEN

83099

Tezin Enstitüye Verildiği Tarih : 11.OCAK.1999
Tezin Savunulduğu Tarih : 03.ŞUBAT.1999

Tez Danışmanı : Doç. Dr. Bülent ÖRENCİK
Diğer Jüri Üyeleri : Prof. Dr. Emre HARMANCI
                      Prof. Dr. Günsel Durusoy

ŞUBAT 1999
PREFA

Cellular communications is one of the fastest growing and most challenging telecommunication applications ever. Today, it represents a large and continuously increasing percentage of all new telephone subscribers around the world. In the long term, cellular digital technology may become the universal way of communication.

The mobile communications market has experienced rapid growth in European Post Offices and Telecommunication (CEPT) Europe. This has been driven by the market forces, technological development, and new forms of cooperation in the areas of standartization and implementation of new systems. A major product of this standards work within CEPT Europe has been the GSM standard. The Global System for Mobile Communication was developed as the next generation digital cellular mobile communication system for CEPT Europe.

One of the application of GSM is remote control. Following chapters tells us what is remote control and how to implement it. All applications which handled by GSM phone, is done by using PC. In the second section, GSM services are introduced. In the third section, the SMS message types and structures are defined. All commands that are defined in GSM 07.07 and 07.05 standard, are given in chapter 4. Chapter 5 tell us how to implement remote control application by using AEG handy.

Special thanks to my friends Deniz Arbatli and Cengiz Gencer.

February 1999

Bülent ÖNEN
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFACE</td>
<td></td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
<td>ix</td>
</tr>
<tr>
<td>SUMMARY</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>ÖZET</td>
<td></td>
<td>xii</td>
</tr>
<tr>
<td>SECTION 1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>INTRODUCTION TO GSM SYSTEM</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>GSM OPERATIONAL REQUIREMENTS</td>
<td>1</td>
</tr>
<tr>
<td>1.3</td>
<td>GSM TECHNICAL REQUIREMENTS</td>
<td>2</td>
</tr>
<tr>
<td>1.4</td>
<td>GSM PROVIDED SERVICES</td>
<td>3</td>
</tr>
<tr>
<td>1.5</td>
<td>GSM ARCHITECTURE</td>
<td>4</td>
</tr>
<tr>
<td>1.6</td>
<td>GSM NETWORK STRUCTURE</td>
<td>6</td>
</tr>
<tr>
<td>1.6.1</td>
<td>Mobile Station</td>
<td>7</td>
</tr>
<tr>
<td>1.6.2</td>
<td>Base Station System</td>
<td>8</td>
</tr>
<tr>
<td>1.7</td>
<td>Remote Control Project</td>
<td>9</td>
</tr>
<tr>
<td>SECTION 2</td>
<td>SERVICES OFFERED BY GSM 07.05 AND GSM 07.07</td>
<td>10</td>
</tr>
<tr>
<td>2.1</td>
<td>CALL CONTROL</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>NETWORK SERVICES</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>MOBILE EQUIPMENT CONTROL AND STATUS COMMANDS</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>SHORT MESSAGE SERVICE (SMS)</td>
<td>12</td>
</tr>
<tr>
<td>SECTION 3</td>
<td>TECHNICAL REALIZATION OF THE SHORT MESSAGE SERVICE (SMS) POINT-TO-POINT (PP)</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>SERVICES AND SERVICE ELEMENTS</td>
<td>14</td>
</tr>
<tr>
<td>3.2</td>
<td>SHORT MESSAGE SERVICE ELEMENTS</td>
<td>16</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Validity Period</td>
<td>16</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Service-Centre-Time-Stamp</td>
<td>16</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Protocol Identifier</td>
<td>16</td>
</tr>
<tr>
<td>3.2.4</td>
<td>More-Message-To-Send</td>
<td>16</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Delivery of Priority and non-Priority Messages</td>
<td>16</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Message-Waiting</td>
<td>17</td>
</tr>
<tr>
<td>3.2.7</td>
<td>Alert-SC</td>
<td>18</td>
</tr>
<tr>
<td>3.2.8</td>
<td>Options Concerning MNRF, MCEF and MWD</td>
<td>18</td>
</tr>
<tr>
<td>3.2.9</td>
<td>Status Report Capabilities</td>
<td>18</td>
</tr>
</tbody>
</table>
SECTION 4 ANALYSIS OF GSM 07.07 AT COMMANDS

4.1 TE - TA INTERFACE COMMANDS
4.1.1 Write Command Line Termination
4.1.2 Write Response Formatting Character
4.1.3 Write Command Line Editing Character
4.1.4 Write Echo Mode
4.1.5 Write Result Code Suppression Mode
4.1.6 Write Result Code Format
4.1.7 Write Connect Result Code Format
4.1.8 Write Data Carrier Detection Mode
4.1.9 Write DTR Mode
4.2 GENERAL COMMANDS
4.2.1 Read Manufacturer Information of ME
4.2.2 Read Model Information of ME
4.2.3 Read Version Information of ME
4.2.4 Read IMEI of ME
4.2.5 Write Character Set
4.2.6 Read Current Character Set
4.2.7 Read Supported Character Sets
4.2.8 Write to Default
4.2.9 Read Complete Capabilities
4.2.10 Write Country Code
4.3 CALL CONTROL COMMANDS
4.3.1 Write Dial Number Type
4.3.2 Read Current Dial Number Type
4.3.3 Read Supported Dial Number Types
4.3.4 Write Call Mode
4.3.5 Read Current Call Mode
4.3.6 Read Supported Call Modes
4.3.7 Terminate Current Call
4.3.8 Read Result of Last Unsuccessful Call
4.3.9 Send Setup Response
4.3.10 Terminate Current Call
4.3.11 Switch to Data Mode
4.3.12 Write Answer Mode
4.3.13 Write Call Terminate Request Timeout
4.3.14 Write Digit Send Timeout
4.3.15 Write Call Termination Timeout
4.4 NETWORK SERVICE RELATED COMMANDS
4.4.1 Write Network Registration Presentation Status
4.4.2 Read Current Network Registration Status
4.4.3 Read Supported Network Registration Presentation Status
4.4.4 Register GSM Network Operator
4.4.5 Read Registered GSM Network Operator
4.4.6 Read Present GSM Network Operators
4.4.7 Write CLIP Status
4.4.8 Read CLIP Status
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Bearer and teleservices.</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>GSM interfaces.</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Remote Control Reference Model.</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>The Short Message Service mobile originated, point-to-point.</td>
<td>15</td>
</tr>
<tr>
<td>3.2</td>
<td>The Short Message Service mobile terminated, point-to-point.</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of how information on one MS can be put in relation to SCs in order to fulfil the requirement of Alert-SC mechanism.</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>Reference configuration of the MS which apply to the SMS</td>
<td>21</td>
</tr>
<tr>
<td>5.1</td>
<td>Local Application Software Access CAF</td>
<td>69</td>
</tr>
<tr>
<td>5.2</td>
<td>Distribution of Events Using the SLAM</td>
<td>72</td>
</tr>
<tr>
<td>5.3</td>
<td>Handling of Request that Complete Asynchronously</td>
<td>73</td>
</tr>
</tbody>
</table>
SUMMARY

Cellular communications is one of the fastest growing and most challenging telecommunication applications ever. Today, it represents a large and continuously increasing percentage of all new telephone subscribers around the world. In the long term, cellular digital technology may become the universal way of communication.

The mobile communications market has experienced rapid growth in European Post Offices and Telecommunication (CEPT) Europe. This has been driven by the market forces, technological development, and new forms of cooperation in the areas of standardization and implementation of new systems. A major product of this standards work within CEPT Europe has been the GSM standard. The Global System for Mobile Communication was developed as the next generation digital cellular mobile communication system for CEPT Europe.

One of the application of GSM is remote control. Following chapters tells us what is remote control and how to implement it. All commands that are defined in GSM 07.07 and 07.05 standard, are given in chapter 4. Chapter 5 tell us how to implement remote control application by using AEG handy.

Services are defined as anything the end user explicitly sees as worth paying for. Services are classified three groups: 1) teleservices, 2) bearer services, and 3) supplementary services.

Teleservices is a type of telecommunication service that provides complete capability, including terminal equipment functions, for communication between users according to established protocols.

Bearer services, on the other hand, is a type of service that provides the capability for the transmission of signals between user network interfaces. Some bearer service are as follows:

- Data service
- Short Message Service
- Cell broadcast
- Local Features

The goal of the project Remote Control is to provide a service interface to a TE enabling the TE to use services of the MS. For offering a generic interface the
project will use the means standardized in GSM 07.07 and GSM 07.05 for implementing this interface. By using a standardised mechanism the interface is open to any application which are able to use this mechanism. AT command set is used by supports MS functionality.

In the approach taken in this document is depicted. First, the services which must be offered by a MS to remote applications are analysed. A remote application is an application using the services offered by MS. The application may e.g. run on a PC connected to the MS by a communication interface. Primarily it states the services offered by GSM 07.05 and GSM 07.07 and then explains the extended AT - Command set describing each command in detail and grouping them according to their functionality's, Second, as a result of this analysis an ideal software architecture has been stated and compared with the existing one in order to point out the changes which have to be implemented.
ÖZET


Yüksek ses kalitesi
Karmaşık bir şebeke yapısında başarılı çağrı kurma
Ekonomik oluşu
ISDN ile integrasyonu


Bu proje bize terminal kullanıcılar cep telefonundan yapılan her işlevin terminalden yapılmasını sağlamaktadır. Terminal tarafından sadece komünikasyon yazılımı bulunmaktadır. Bu yazılım VT220 emulasyon programı veya windows programındaki hiper terminal olabilir. Terminalle cep telefonu arasında seri bağlantılı kablosu kullanılmaktadır. Terminalden girilen komutlar at karakterleri ile başlamak zorundadır. Temel ve genişletilmiş komutlar olmak üzere iki komut kümesi vardır. Temel komutlar at ile, genişletilmiş komutlar ise at+ ile başlamak zorundadır. Terminal kullanılarak yerine getirilen GSM uygulamaları şunlardır:

Mesaj Servisleri : Kısa mesaj gönderme, alma, silme, yazma.
Yayın Mesaj Servisi : Şebeke tarafından gönderilen mesajların alınması.
Telefon Rehberi Uygulamaları : Telefonka kayıt yapma, kayıt değiştirme ve silme.
Şebeke Uygulamaları : Şebekeye erişim ,şebekele listelenmesi, değiştirilmesi, kaydının silinmesi.
Güvenlik işlemlerleri : Şifre değiştirme.
Pil Durumunun Gözlenmesi.
Veri Servisi: FAX gönderilmesi ve alınması.

Bu projede, veri servisi dışındaki tüm servisler tasarlanmıştır.

Yukarıda verilen servisler terminalden GSM 07.05 ve GSM 07.07 standartlarında tanımlanan komutlar yardımcıyla yerine getirilir. GSM 07.05 'de mesaj servislerine ilişkin komutlar verilmektedir. GSM 07.07 'de diğer servislere ilişkin komutlar tanıtılmaktadır.


Cep telefoni terminale seri portu üzerinden bağlandığındda işletim sistemi ATI task ini adılı fonksiyonu çağırılmaktadır. Bu fonksiyonda ATI modülünün bellegini başlangıç değerleriyile doldurmakta mesaj kuyruğunu yaratmakta ve işletim sistemine bundan sonra çağracağı fonksiyonun adresini (ati_task) göndermektedir. ATI modülünün başlangıcı bu şekilde yapılmaktadır. Bundan sonra ATI modülüne bir mesaj gelirse işletim sistemi atı_task fonksiyonunu çağırılmaktadır. ATI modülü diğer modüllerle ilkelleri vasatıyla haberleşmektedir. İlkel okuma ve yazma mesaj kuyruğu üzerinden yapılmaktadır.

ATI modülünün işlemleri:

Seri iletişim protokollünün sağlanması
Gelen karakter katarının V.25 protokolüne uygunluğunun kontrolü
Birden fazla karakter setinin desteklenmesi
Alınan karakterlerin komutlara ayrılanıştırılması
Komut parametrelerinin belirlenmesi
Komut tablosundan girilen komutla ilgili fonksiyonun belirlenmesi ve bu fonksiyonun çağrılması
Fonksiyonda parametre hata kontrolünün yapılması
Komutla ilgili işlem yerine getirilmesi
Kuyruktan mesaj okuma ve kuyruğa mesaj yazma

Bazı komutlarda diğer modüllerde ATI modülünün iletişimi gerçekleştirildir. Bu durumda ATI modülü ilgili modüle ilkeler gönderilir ve ilgili modülden ilkeler beklemektedir. ATI modülün mesaj kuyruğuna ilgili ilkelin cevabı geleince işletim sistemi ATI modülü çağırılmaktar. ATI modülü de ilgili ilkeleri kuyruktan çekmekte ve komutun cevabını hazırlayap seri porttan terminalde göndermektedir. Komutların ve karakterlerin zaman aşımı olup olmadığını belirlenmesi ATI modülü tarafından yapılmaktadır.

Tezin birinci bölümünde, GSM sistemleri tanıtılmaktadır. GSM sistemlerine giriş yapılakta, teknik özellikleri tanıtılmakta, şebekeleri aprşı açıklanmaktadır. Birinci bölümün son kısmında tezde gerçekleştirilen projede kısaltı olarak tanıtılmıştır.

İkinci bölümde, GSM 07.05 ve GSM 07.07 standartlarında verilen GSM servisleri verilmektedir. Bunlar:

Çağrı kontrolü: Çağrı kurulması, çağrı sonlandırılması, çağrı özellikleri yonelik servislerdir. Çağrı özellikleri çağrı yönlendirme, bekletme, üçlu konuşma, çağrı transferi, uyanırma, rahatsız edilmeme özellikleri.

Şebekeler servisleri: Şebekeler seçme, listeleme ve değiştirmek imkanı sağlayan hizmetlerdir.


Cep telefonu kontrol ve durum sorgulama servisleri: telefonun durumunun belirlenmesi (zil calması, SIM kartını takılı olup olmama, saat ve tarih bilgilerinin değiştirilmesi, alarm durumunun sorgulanması.

Üçüncü bölümde, Kısa mesaj formatı, yapısı tanıtılmaktadır. Bu mesajlar, cep telefonuya şebekenin arasında gönderilir mesajlardır. Şebekeden telefona, telefondan şebekteye gönderilir veri ve durum kontrol mesajlarının yapısı bu bölümde ayrıntılı olarak anlatılmaktadır. Şebekeden cep telefonuna gönderilir ve cep telefonunun şebekte gönderilen mesajların yapısı bu bölümde anlatılmıştır.

Dördüncü bölümde terminalden girilecek komutlar ve bu komutların formatı tanıtılmaktadır. Bu projede, dördüncü bölümde listelenen tüm komutlar tasarılanmıştır.

Terminal telefon arabirim komutları: Terminalin karakter özelliklerini öğrenme ve değiştirme yönelik komutlardır. Bunlar: Komut sonu karakterini değiştirme, yansıtma yapma komutu, sonuç kodu formatı seçme, sonuç kodu sıkıştırma, cevap format karakteri seçme komutlarıdır.

Genel komutlar: Terminal karakter setini öğrenme, değiştirme, listeleme komutları, telefonun model bilgisini öğrenme, versiyon numarasını öğrenme komutlarıdır.

Çağrı kontrol komutları: Çağrı kurma, cevap verme, çağrı sonlandırma komutlarıdır.

Şebekeler servislerine ilişkin komutlar: Bunlar cep telefonunun bağlı bulunduğu şebekeyi öğrenmeyi, değiştirmeye yönelik komutlardır. Bunlar şebeke listeleme, şebeke değiştirme, şebeke silme komutlarıdır.

Cep telefonu kontrol ve durum komutları: Cep telefonunun şifrelerinin değiştirilmesi, pil durumunun öğrenilmesi, şebekeyden alınan işaretin gücünün öğrenilmesi, telefon rehberi seçme, telefon defterine kayıt yapma, rehberdeki kayıtları değiştirme, kayıtları bulma komutlarıdır.

Genel konfigürasyon komutları: Bunlar kısa mesaj servisi için şebeke seçme ve mesaj formatlarını değiştirmeye yönelik komutlardır.


SECTION 1

INTRODUCTION

1.1 INTRODUCTION TO GSM SYSTEM

Cellular telecommunications is one of the fastest growing and most challenging telecommunication applications ever. Today, it represents a large and continuously increasing percentage of all new telephone subscribers around the world. The mobile communications market has experienced rapid growth in European Post Offices and Telecommunication (CEPT). This has been driven by market forces, technological and development, and new forms of cooperation in the areas of new system standardisation and implementation. A major product of this standards work within CEPT Europe has been the Global System for Mobile Communication (GSM) standard. The GSM was developed as the next-generation digital cellular mobile communication system for CEPT Europe. The standardisation work for the first implementation in 1991 was completed in early 1990.

1.2 GSM OPERATIONAL REQUIREMENTS

A list of operational requirements was developed that consisted of the following.

- High audio quality and link integrity;
- High spectral efficiency;
- Identical system in all countries;
- Intersystem roaming (international roaming needs standardised air interface);
- High degree of flexibility (open architecture that will allow new services to be introduced at a future date);
- Economy in both sparsely and heavily populated areas;
- Integration with ISDN;
Other security features;  
A range of additional features; such as short message service and use of facsimile system;  
Easy to introduce the system;  
Low - cost infrastructure

1.3 GSM TECHNICAL REQUIREMENTS

GSM uses both TDMA and FDMA to transmit and recover information. These system use data packets at specific times at specific frequencies. Thus, several conversations take place simultaneously and at the same frequency using different time slots. Systems are also frequency duplex so that the transmit and receive frequencies are different, and both sides of the transmission (Mobile-to-Base and Base-to-Mobile) are concurrent.

The spacing between the carries in GSM system is 200 kHz. Eight time slots carry speech and data in a GSM system. The bandwidth for the GSM system is 25 MHz, which provides 125 carriers each having a bandwidth of 200 kHz. Due to interference to the other systems, the very first carrier is not used, thus reducing the number of carriers to 124. With eight users per channel there are about 1.000 actual speech or data channels. The number of channels will double to about 2.000 as the half rate speech coder is introduced. The frequency band used for the unlink is 890 Mho to 915 Mho (from MS to base station) and for the downland 935 Mho to 960 Mho (from base station to MS).

The modulation method in GMS is Gossan Minimum Shift Keying (GMS), which facilitates the use of narrow bandwidth and coherent detection capability. In GMS the rectangular pulses are passed through a Gossan filter prior to their passing through a modulator. This modulation scheme almost satisfies the adjacent channel power spectrum density requirement of -60dBc CCIR. The normalised pre-Gaussian bandwidth is kept at 0.3, which corresponds to a filter bandwidth of 81.25 kHz for an aggregate data rate of 270.8 Kbps. With 200 kHz of carrier spacing and this data rate the spectral efficiency of the system is 1.35 b/s/Hz (270.8/200). With the bit interval of 3.7 Ms, the GSM signal will encounter significant intersymbol interference in the mobile radio path due to multipath (multipath minimal delay spread 3 ms to 6 ms in urban areas). As a consequence, an adaptive equaliser is used. There are eight time slots in a frame and 26 or 51 frames in a multiframe.

With 270.8 Kbps divided among eight users in GSM, the per user data rate is 33.85 Kbps. The speech coder is a regular pulse excitation with long term predictor for full rate speech that converts speech to 13 Kbps. In the near future, a half rate
coding scheme at a rate of roughly 7 Kbps will be used. The data transmission rates use 12 Kbps, 6 Kbps, 4.8 Kbps, and 2.4 Kbps rates, respectively, plus the control bits with each of these. Each base station is equipped with a certain number preassigned carrier frequencies.

There are five different categories of mobile telephone units specified for the European GSM system: 0.8W, 2W, 5W, 8W, and 20W. The power level can be adjusted to vary between 3.7 mW to 20W. To optimize cochannel interference, each BS individually directs MS to use the minimum power setting that is necessary for reliable transmission. The setting is determined by BS and provided to the MS. The GSM air interface allows for frequencies to be hopped to prevent multipath problems resulting in excessive bit error rates. Both the mobile and the base station will use Discontinuous Transmission (DTx). This will allow the mobile to save the battery life and base station to reduce cochannel interference.

1.4 GSM PROVIDED SERVICES

Services are defined as anything the end user explicitly sees as worth paying for. Services are classified into three groups: 1) teleservices, 2) bearer services, and 3) supplementary services.

Teleservices is type of telecommunication service that provides complete capability, including terminal equipment functions, for communication between users according to established protocols.

Bearer services, on the other hand, is a type of service that provides the capability for the transmission of signals between user network interfaces. Some bearer services are as follows.

- Data services;
- Short message services (SMS);
- Cell broadcast;
- Local features.

Supplementary services are defined as add-ons, that is additional features to both teleservices and bearer services. In a way, these are value-added services on top of teleservices and bearer services. Figure 1.1 provides access points for bearer and teleservices.
1.5 GSM ARCHITECTURE

A GSM system is basically designed as a combination of three major subsystems: the network subsystem, the radio subsystem, and the operation support system. In order to ensure that network operators will have several sources of cellular infrastructure equipment, GSM decided to specify not only air interface, but also the main interfaces that identity different parts. There are three dominant interfaces, namely, an interface between MSC and the Base Transceiver Station (BTS), and an Um interface between the BTS and MS. These three interfaces are shown in Figure 1.2.
The network subsystem includes the equipment and functions related to end-to-end calls, management of subscribers, mobility, and interface with the fixed PSTN. In particular, the switching subsystem consists of MSCs, Visitor Location Register (VLR), Home Location Register (HLR), Authentication Centre (AUC), and Equipment Identity Register (EIR). The MSC provides call setup, routing, and handover between BSCs in its own area and to/from other MSC; an interface to the fixed PSTN; and other functions such as billing. The HLR is a centralized database of all subscribers registered in a PLMN. There may be more than one HLR in PLMN, but the individual subscriber has entry to only one of them. The VLR is a database of all mobile, currently roaming in the MSCs area of control. As soon as MS roams into a new MSC area, the VLR connected to that MSC will request data about the MS from the HLR. At the same time, HLR will be informed as to which MSC area the MS resides. If, at a later time, MS wants to make a call, the VLR will have all the information needed for the call setup without having to interrogate the HLR each time. Thus, VLR in one sense is a distributed HLR. VLR also contains more exact information about the mobile location. The AUC is connected to the HLR. The function of the AUC is to provide HLR with authentication parameters and ciphering keys that are used for security purposes. The EIR is the database where the International Mobile Equipment Identity (IMEI) numbers for all registered mobile equipment are stored.

Some other components of the network are Echo Canceller, which reduces the annoying effect caused by the network when connected to a PSTN circuit; and the network Interworking Function (IWF), which is the interface between MSC and the other networks such as PSDN and ISDN.

The radio subsystem includes the equipment and functions related to the management of the connections on the radio path, including the management of handovers. It mainly consist of a BSC, BTS, and the MS. MS is traditionally listed as a part of the radio subsystem even though it is always one end of the conversational path and holds dialogues with the network subsystem for the management of its mobility. The MS includes both the capabilities of network termination and user termination. The GSM system is realized as a network of radio cells, which together provide complete coverage of the service area. Each cell has a BTS with several transceivers. A group of BTSs are controlled by one BSC. There are various configurations of BSC-BTS. Some configurations are best suited for high traffic, and some are meant to serve moderate-to-low traffic areas. A BSC together are known as a BSS, which is viewed by the MSC through a single interface as being the entity responsible for communication with MSs in a certain area. A BSS is associated with the radio channel management, transmission functions, radio link
control, and quality assessment and preparation for handover. BSS ensures the coverage of $N$ cells, where $N$ can be one or more.

The Operational and Maintenance Center (OMC) subsystem includes the operation and maintenance of GSM equipment and supports the operator network interface. It is connected to all equipment in the switching system and to the BSC. OMC performs GSM's administrative functions (for example, billing) within a country. One of the OMC's most important functions is the maintenance of the country's HLR. Depending upon the network size, each country may have more than one OMC. The global and centralized management of the network is provided by the Network Management Center, while the OMC is responsible for the regional management of the network.

1.6 GSM NETWORK STRUCTURE

Every telephone network needs a well-designed structure in order to route incoming calls to the correct exchange and finally to the called subscriber. In a mobile network, this structure is of great importance because of the mobility of all its subscribers. In the GSM system, the network is divided into the following partitioned areas.

- GSM service area;
- PLMN service area;
- MSC service area;
- Location area;
- Cells.

The GSM service area is the total area served by the combination of all member countries where a mobile can be serviced. The next level is the PLMN service area. There can be several within a country, based on its size. The links between a GSM/PLMN network and other PSTN, ISDN, or PLMN networks will be on the level of international or national transit exchanges. All incoming calls for a GSM/PLMN network, will be routed to a Gateway MSC. In a GSM/PLMN network, all mobile terminated calls will be routed a Gateway MSC. Call connections between PLMNs, or to fixed networks, must be routed through certain designated MSCs called a gateway MSC. The gateway MSC contains the interworking functions to make these connections. They also route incoming calls to the proper MSC within the network. The next level of division is the MSC/VLR service area. In one PLMN there can be several MSC/VLR service areas. MSC/VLR is a sole
controller of calls. In order to route a call to mobile subscriber, the path through the network links to the MSC in the MSC area where the subscriber is currently located. The mobile location can be uniquely identified since the MS is registered in a VLR, which is generally associated with an MSC.

There are several LA s within one MSC/VLR combination. A LA is a part of the MSC/VLR service area in which a MS may move freely without updating location information to the MSC/VLR exchange that controls the LA. Within a LA a paging message is broadcast in order to find the called mobile subscriber. The LA can be identified by the system using the Location Area Identity (LAI). The LA is used by the GSM system to search for a subscriber in active state. A LA is divided into many cells. A cell is an identity served by one BTS. The MS distinguishes between cells using the Base Station Identification Code (BSIC) that the cell site broadcasts over the air.

1.6.1 Mobile Station

The MS includes radio equipment and the man machine interface (MMI) that a subscriber needs in order to access the services provided by the GSM PLMN. MSs can be installed in vehicles or can be portable or handheld stations. The MS may include provisions for data communications as well as voice. A mobile transmits and receives messages to and from the GSM system over the air interface to establish and continue connections through the system. Each MS is identified by an IMEI that is permanently stored in the mobile MSC. The IMEI can be used to identify mobile units that are reported stolen or operating incorrectly. Different subscriber identities are used in different phases of call setup. The Mobile Subscriber ISDN Number (MSISDN) is the number that the calling party dials in order to reach the subscriber. It is used by the land network to route calls toward an appropriate MSC. The international Mobile subscriber Identity (IMSI) is the primary function of the subscriber within the mobile network and is permanently assigned to him. The GSM system can also assign Temporary Mobile Subscriber Identity (TMSI) to identify a mobile. This number can be periodically changed by the system and protects the subscriber from being identified by those attempting to monitor the radio channels.

By making a distinction between the subscriber identity and the mobile equipment identity, a GSM PLMN can route calls and perform billing based on the identity of the subscriber rather than the mobile unit being used. This can be done using a removable Subscriber Identity Module (SIM). A smart card (SC) is one possible implementation of a SIM; the other implementation can be the module mounted on the mobile equipment. The TMSI pertaining to the identity of the
subscriber is stored in the SIM module itself. When the SIM is in the mobile unit, a location update procedure registers the subscriber’s new location, allowing proper routing of incoming calls.

Functions of MS:

Voice and data transmission;
Frequency and time synchronization;
Monitoring of power and signal quality of the surrounding cells for optimum handover;
Provision of location updates;
Equalization of multipath distortions;
Display of short messages up to 160 characters long;
Timing advance.

1.6.2 Base Station System

The BSS is set of BS equipment (such as transceivers and controllers) that is in view by the MSC through a single A interface as being the entity responsible for communicating with MSs in certain area. The radio equipment of a BSS may be composed of one or more cells. A BSS may consist of one or more BSs. The interface between BSC and BTS is designed as an A-bis interface. The BSS includes two types of machines: the BTS in contact with the MSs through the radio interface and the BSC, the latter being in contact with the MSC. The function split is basically between transmission equipment, the BTS, and a managing equipment at the BSC. A BTS comprises radio transmission and reception devices, up to and including the antennas, and also all the signal processing specific to the radio interface. A single transceiver within BTS supports eight basic radio channels of the same TDMA frame. A BSC is a network component in the PLMN that functions for control of one or more BTS. It is functional entity that handles common control functions within a BTS.

A BTS is a network component that serves one cell and is controlled by a BS. BTS is typically able to handle three to five radio carriers, carrying between 24 and 40 simultaneous communications. Reducing the BTS volume is important to keeping down the cost of the cell sites.
1.7 Remote Control Project

The goal of the project Remote Control is to provide a service interface to a TE enabling the TE to use services of the MS. For offering a generic interface the project will use the means standardized in GSM 07.07 and GSM 07.05 for implementing this interface. By using a standardised mechanism the interface is open to any application which are able to use this mechanism. AT command set is used by supports MS functionality challenging.

In the approach taken in this document is depicted. First, the services which must be offered by a MS to remote applications are analysed. A remote application is an application using the services offered by MS. The application may e.g. run on a PC connected to the MS by a communication interface. Primarily it states the services offered by GSM 07.05 and GSM 07.07 and then explains the extended AT-Command set describing each command in detail and grouping them according to their functionality's. Second, as a result of this analysis an ideal software architecture has been stated and compared with the existing one in order to point out the changes which have to be implemented.

The discussion is based on the reference mode in shown in Figure 1.3 comprising a TE (e.g. a computer) and a ME interfaced by a TA. The model has the advantage that it allows any physical implementation:

![Figure 1.3 Remote Control Reference Model](image)

- TA, ME and TE as three separate entities
- TA integrated under ME cover, and the TE implemented as a separate entity
- TA and ME integrated under the TE cover as a single entity

The connection between TE and TA is a serial connection, e.g. according to the V.24 standard.
SECTION 2

SERVICES OFFERED BY GSM 07.05 AND GSM 07.07

This chapter gives a short introduction into the commands offered by GSM 07.07 and GSM 07.05. The description is organised according to functional groups. A functional group is a set of commands related to a single topic. The following functional groups were identified:

- Call Control
- Network Services
- Mobile Equipment Control and Status Commands
- Short Message Service (point to point and cell - broadcast)

2.1 CALL CONTROL

In GSM 07.07, there is a number commands defined for call control. These commands relate to call mode setting (voice, data), call - setup and release, and to controlling an active call and a call on hold. In detail, there are commands for:

setting up a call; the following parameters of a call can be controlled:

- mode (voice, data)
- bearer service
- speed (data calls only)
- connection element (transparent, non-transparent)
- RLP parameters
- phone number (by name, direct number, phonebok&entry number)
- type of address
- suppress CLIR
- calling within a closed user group
- releasing calls
- alternating between hold and active calls
2.2 NETWORK SERVICES

In GSM 07.07 there is a number of commands defined related to network services. These commands include commands for supplementary services handling, MSISDN query, ME and network facility locking, and network registration query. In detail, the commands control the following services:

- getting subscriber’s MSISDNs
- indicating change in network registration status
- locking network facilities
- changing password
- calling line identification presentation (CLIP)
- calling line identification restriction (CLIR)
- connected line identification presentation (COLP)
- closed user group (CUG)
- call forwarding (CF)
- call waiting (CW)
- call hold (HOLD)
- call transfer

2.3 MOBILE EQUIPMENT CONTROL AND STATUS COMMANDS

Mobile equipment control and status commands include commands include for handling the ME keypad, display and indicators. In detail these commands are:

- query activity status of ME (ringing, idle, etc)
- set phone functionality (levels may be manufacturer specific)
- enter pin
- query battery connection status and battery level
- query signal quality
- controlling access to ME keypad and display (access only by ME, by TE, or by both)
- controlling the setting of the indicators in ME
- set clock
- set alarm time
- generic SIM access

A second set of control commands is dedicated to managing the phonebook in the MS. In detail these commands are:
• select phonebook memory storage (SIM, ME internal)
• read phonebook entries
• find phonebook entries
• write phonebook entry

2.4 SHORT MESSAGE SERVICE (SMS)

In three interface protocols for controlling the short message service from a TE are specified. These are
• the Block Mode Protocol
• the Text Mode Protocol
• the PDU Mode Protocol

The Block Mode Protocol because it does not build on AT commands. This protocol is not considered in the following. The Text Mode Protocol is suitable for application software built on command structures like those defined in V.25 Ter. The encoding of the message blocks and the marshalling of parameters into messages has to be done in the ME. Commands, parameters, and message blocks are transmitted as characters. The PDU Mode Protocol is also a character based protocol but in contrast to the Block Mode Protocol the transferred message blocks are binary encoded. This protocol is of use if the encoding of the message blocks is done in the TE and the ME transfers them transparently to the network.

The following commands are specified for the short message service point to point and cell - broadcast:

• query supported service (mo, mt)
• select cell broadcast message type
• select preferred message storage (SIM, ME)
• select protocol used (Block, Text, or PDU Mode)
• set service centre address
• set / show text mode parameters (VP, PID, DCS, FO)
• save settings
• restore settings
• indication of new incoming message
• list messages from preferred message storage
• read specific message from the preferred message storage
• send SMS - SUBMIT
• write message to message storage
- delete message from message storage
- send SMS - COMMAND
- select CB message types.

The PDU protocol mode uses the same commands as the Text Protocol mode. However, some commands and responses have a different format.
SECTION 3

TECHNICAL REALIZATION OF THE SHORT MESSAGE SERVICE (SMS) POINT-TO-POINT (PP)

The Point-to-Point Short Message Service (SMS) provides a means of sending Service Centre, which acts as a store and forward centre for short messages. Thus GSM PLMN needs to support the transfer of short messages between Service Centres and mobiles.

Two different point-to-point services have been defined: mobile originated and mobile terminated. Mobile originated messages will be transported from an MS to a Service Centre. These may be destined for other mobile users, or for subscribers on a fixed network. Mobile terminated messages will be transported from a Service Centre to an MS. These may be input to the Service Centre by other mobile users (via a mobile originated short message) or by a variety of other sources, e.g. speech, telex, or facsimile.

The European Telecommunications Standard (ETS) describes the point-to-point Short Message Service (SMS) of the GSM PLMN system. It defines:

- the services and service elements;
- the network architecture;
- the service functionality;
- the MSC functionality;
- the routing requirements;
- the protocols and protocol layering;

3.1 SERVICES AND SERVICE ELEMENTS

The SMS provides a means to transfer short messages between a GSM MS and SME via an SC. The SC serves as an interworking and relaying function of the message transfer between the MS and the SME.

The short message point-to-point services comprise two services:

- SM MT (Short Message Mobile Terminated Point-to-Point);
- SM MO (Short Message Mobile Originated Point-to-Point).

SM MT denotes the capability of the GSM system to transfer a short messages submitted from the SC to one MS, and to provide information about the delivery of the short message either by a delivery report or a failure report with a specific mechanism for later delivery; see Figure 3.1
SM MO denotes the capability of the GSM system to transfer a short message submitted by the MS to one SME via an SC, and to provide information about the delivery of the short message either by a delivery report or a failure report. The message must include the address of that SME to which the SC shall eventually attempt to relay the short message; see Figure 3.2. The text message to be transferred by means of the SM MT or SM MO contain up to 140 octets.

![Short message delivery](image)

Figure 3.1. The Short Message Service mobile originated, point-to-point.

![Short message submission](image)

Figure 3.2. The Short Message Service mobile terminated, point-to-point.

An active MS be able to receive a short message TPDU (SMS DELIVER) at any time, independently of whether or not there is a speech or data call in progress. A report will always be returned to the SC; either confirming that the MS has received the short message, or informing the SC that it was impossible to deliver the short message TPDU to the short message TPDU to the MS, including the reason why.

An active MS shall be able to submit a short message TPDU (SMS-SUBMIT) at any time, independently of whether or not there is a speech or data call in progress. A report will always be returned to the MS; either confirming that the SC has received the short message TPDU, or informing the MS that it was impossible to deliver the short message TPDU to the SC, including the reason why.

When the transmission or reception of a short message coincide with a change of state in the MS, i.e from busy to idle or from idle to busy, or during a handover, the short message transfer might be aborted.

It is also possible for two short messages to be received in sequence having the same originating address and identification, i.e message reference number (MO) or (SC) Timestamp (MT). Such a situation may be due to errors at the RP or CP layers (e.g. during inter MSC handover) where it may be duplicated message or otherwise it may be a valid new message. The receiving entity should therefore make provision to check other parameters contained in the short message to decide whether the second short message is to be discarded.
3.2 SHORT MESSAGE SERVICE ELEMENTS

The SMS comprises 7 elements particular to the submission and reception of messages:

validity period;
service centre time stamp
protocol identifier
more message to send
priority
message waiting
alert SC.

3.2.1 Validity Period

The Validity period is the information element which gives an MS submitting an SMS-SUBMIT to the SC the possibility to include a specific time period value in the short message (TP-Validity-Period). The TP-Validity-Period parameter value indicates the time period for which the short message is valid, i.e. for how long the SC shall guarantee its existence in the SC memory before delivery to the recipient has been carried out.

3.2.2 Service-Centre-Time-Stamp

The Service-Centre-Time-Stamp is the information element by which the SC informs the recipient MS about the time of arrival of the short message at the SM-TL entity of the SC. The time value is included in every SMS-DELIVER (TP-Service-Centre-Time-Stamp Field) being delivered to the MS.

3.2.3 Protocol Identifier

The Protocol-Identifier is the information element by which the SM-TL either refers to the higher layer protocol being used, or indicates interworking with certain type of telematic device. The Protocol-Identifier information element makes use of a particular field in the message types SMS-SUBMIT, SMS-DELIVER, and SMS-COMMAND TP-Protocol-Identifier(TP-PID).

3.2.4 More-Message-To-Send

The More-Message-To-Send is the information element by which the SC informs the MS that there is one or more messages waiting in that SC to be delivered to the MS. The More-Message-to-Send information element makes use of a boolean parameter in the message SMS-DELIVER, TP-More-Message-to-Send (TP-MMS).

3.2.5 Delivery of Priority and non-Priority Messages

Priority is the information element provided by an SC or SME to indicate to the PLMN whether or not a message is a priority message.
Delivery of a non-priority message will not be attempted if the MS has been identified as temporarily absent.

Delivery of a non-priority message will be attempted if the MS has not been identified as temporarily absent irrespective of whether the MS has been identified as having no free memory capacity.

Delivery of a priority message will be attempted irrespective of whether or not the MS has been identified as temporarily absent, or having no free memory capacity.

3.2.6 Message-Waiting

The Messages-Waiting is the service element that enables the PLMN to provide the HLR and VLR with which the recipient MS is associated with the information that there is a message in the originating SC waiting to be delivered to the MS. The service element is only used in case of previous unsuccessful delivery attempts due to temporarily absent mobile or MS memory capacity exceeded. This information denoted the Message-Waiting-Indication (MWI), consist of Message-Waiting-Data (MWD), the Mobile-Station-Not-Reachable-Flag (MNRF) and Mobile-Station-Memory-Capacity-Exceeded-Flag (MCEF) located in the HLR, and the Mobile-Station-Not-Reachable-Flag located in the VLR. Figure 3.3 shows an example.

HLR;

![Diagram of HLR]

VLR;

![Diagram of VLR]

Figure 3.3. Example of how information on one MS can be put in relation to SCs in order to fulfil the requirement of Alert-SC mechanism.
The MWD shall contain a list of addresses (SC-Addr) of SCs which have made previous unsuccessful delivery attempts of a message. In order to be able to send alert messages to every SC which has made unsuccessful delivery attempts to an MS, the HLR shall store the MSIsdn-Alert together with references to the SC addresses.

The Mobile-Station-Memory-Capacity-Exceeded-Flag (MCEF) within the HLR is a boolean parameter with the value TRUE when the list MWD contains one or more list elements because an attempt to deliver a short message to an MS failed with a cause of MS Memory Capacity Exceeded, and with the value FALSE otherwise.

The Mobile-Station-Not-Reachable-Flag (MNRF) within the HLR and the VLR is a boolean parameter with the value TRUE when the list MWD contains one or more list elements because an attempt to deliver a short message to an MS has failed with a cause of Absent Subscriber, and with the value FALSE otherwise.

3.2.7 Alert-SC

The Alert-SC is the service element, which may be provided by some GSM PLMNs, to inform the SC that an MS

1) to which a delivery attempt has failed because the MS is not reachable or the MS memory capacity was exceeded;

and

2) which is now recognised by the PLMN:

a) to have resumed operation (e.g. to have responded to a paging request); or

b) to have memory newly available (which implies that the mobile is reachable)

is again ready to receive one or more short messages. The SC may - on reception of an Alert-SC - initiate the delivery attempt procedure for the queued messages destined for this MS. To each MS there may be allocated several MSIsdns. When the HLR is to alert an SC that an MS is again attainable it will use a specific MSIsdn value for this purpose; in this specification called MSIsdn-Alert.

3.2.8 Options Concerning MNRF, MCEF and MWD

Setting the Mobile-Station-Not-Reachable-Flag (MNRF) in the VLR is mandatory. It is also mandatory for the VLR to send the 'MS Present' to the HLR when the MS has been detected as becoming active and then to clear MWF.

The Message-Waiting-Data (MWD, the Mobile-Station-Not-Reachable-Flag (MNRF) and the Mobile-Station-Memory-Capacity-Exceeded-Flag (MCEF)) within the HLR are optional. This is linked to the transmission of the 'Alert SC' message.

3.2.9 Status Report Capabilities

The SMS also offers to the SC the capabilities of informing the MS of the status of a previously sent mobile originated short message. The status of the message can be:
Successfully delivered to the SME;

The SC was not able to forward the message to the SME. The reason can be an error of permanent or temporary nature. Permanent errors can be e.g. validity period expired, invalid SME address. Errors of temporary nature can be e.g. SC-SME connection being down, SME temporarily unavailable.

This is achieved by the SC returning a status report TPDU (SMS-STATUS-REPORT) to the originating MS when the SC has concluded the status of the short message. The status report may be initiated by a status port request within mobile originated short message. The status report TPDU is treated as an SMS-DELIVER TPDU by the SC when it comes to delivery procedures e.g. the alerting mechanism. The SC may also return to a non-MS SME the status of a mobile terminated short message. The status report capabilities of the SMS are optional, i.e. the choice of whether to offer status report or not is left to the SC operator.

3.2.10 Reply Path

Reply Path provides a way of both requesting and indicating a service centre's commitment to deliver a reply from the replying MS to the originating SME.

3.3 SERVICE CENTRE AND PLMN INTERCONNECTION

This section deals with the SC only with regard to the interchange of message between SC and MS. Only the requirements put upon the SC by SMS functionality are specified in this section.

3.3.1 Service Centre Connection

One SC may be connected to several PLMNs, and may be connected to several MSCs (SMS-GMSCs or SMS-IWMSCs) within one and the same PLMN. The SC is addressed from the mobile by an E.164 number in the numbering plan of the PLMN to which the SC is connected. This E.164 number shall uniquely identify the SC to that PLMN.

There may be an intermediate network between the PLMN and the SC; in this case the PLMN must autonomously make a connection to the SC using the SC address in this intermediate network.

No mandatory protocol between the SC and the MSC below the transfer layer is specified by GSM; this is a matter for agreement between SC and PLMN operators.

3.3.2 Routing Requirements

3.3.2.1 Mobile terminated short message

The SC sends the short message to the SMS-GMSC. The SMS-GMSC interrogates the HLR to retrieve routing information necessary to forward the short message, and then sends the message to the relevant MSC, transiting other networks if necessary. The MSC then sends the short message to the MS.
3.3.2.2 Mobile originated short message

The MS sends the short message to the MSC. The MS will always address the required SC by an E.164 address. The visited PLMN will route the message to the appropriate SMS-IWMSC in the SCs PLMN, transiting other networks if necessary.

3.4 SERVICE CENTRE FUNCTIONALITY

3.4.1 Service Centre Capabilities

The SC should be capable of

- submitting a short message to an MS, retaining the responsibility of the message until
  1) the report has been received; or
  2) the Validity-Period expires.
- receiving a report from the PLMN;
- receiving a short message from an MS;
- returning a report to the PLMN for a previously received short message.

3.4.2. SC Functional Requirements

The following functional requirements are mandatory for all SCs in order to support the SM-TP towards the PLMN:

1) To identify each SMS-DELIVER sent to an MS in unique way, a time stamp value is included in the field TP-Service-Centre-Time-Stamp, TP-SCTS, of the SMS-DELIVER. The time stamp gives the time when the message arrived at the SC with the accuracy of a second. If two or more messages to the same MS arrive at the within one second, the SC shall modify the time stamp of those message in such a way that
   a) all messages to the MS contain different time stamps;
   b) the modification of the time stamps is kept to a minimum.

2) The SC is only allowed to have one outstanding SMS-DELIVER to a specific MS at a given time.

3) The SC shall be able to initiate overwriting of short messages previously received by the SC if requested by the same originating address (MS or any other source) by use of the same message type.
3.5 MS FUNCTIONALITY

In this section, only the MS functionality related to the short message point-to-point service between the SC and the MS is specified.

3.5.1 MS Capabilities

The MS, when equipped for SMS, should be capable of

- submitting a short message TPDU to an SC, retaining the responsibility of the message until:
  1) the report arrives from the network, or
  2) a timer expires.
- receiving a short message TPDU from an SC;
- returning a delivery report to the network for a previously received short message;
- receiving a report from the network;
- notifying the network when it has memory capacity available to receive one or more short messages when it has previously rejected a short message because its memory capacity was exceeded;
- notifying the SC when a short message is intended to replace a short message the MS has previously submitted to the same destination address.

3.5.2 MS Configuration

The reference configuration is assumed as in Fig. 3.4 i.e only the case where the terminal is integrated in the MS is considered.

![Figure 3.4](image)

Figure 3.4. Reference configuration of the MS which apply to the SMS.

3.6 MSC FUNCTIONALITY

The overall requirements to the MSC with respect to handling of the short message service point-to-point is to cater for the routing and necessary intermediate buffering of the short messages.

3.6.1 MSC Functionality Related to SM-TL

3.6.1.1 Functionality of the SMS-GMSC

When receiving a short message TPDU from the SC, the SMS-GMSC is responsible for the following operations:
- reception of the short message TPDU,
- inspection of the parameters,

if parameters are incorrect:
- returning the appropriate error information to the SC in a failure report.

if errors are not found within parameters:
- interrogating the HLR; retrieving routing information or possible error information,

if HLR is returning error information:
- returning the appropriate error information to the SC in a failure report.

if no errors are indicated by the HLR:
- transferring the short message TPDU to the MSC using the routing the information obtained from the HLR.

When receiving the report associated with the short message from the MSC, the SMS-GMSC is responsible for the following operations:

if the report indicates successful delivery:
- notifying the HLR of the successful delivery, which will cause the HLR to alert any service centres whose addresses are stored in the MWD for the MS.

if the report is a failure report indicating 'absent subscriber'
- requesting the HLR to insert the address of the originating SC into the MWD with cause Absent Subscriber
- establishing, where necessary, a link with the addressed SC
- creating and sending the report to the SC.

if the report is a failure report indicating 'MS capacity exceeded':
- requesting the HLR to insert the address of the originating SC into the MWD with cause MS Memory Capacity Exceeded
- establishing, where necessary, a link with the addressed SSC
- creating and sending the report to the SC.

3.6.1.2 Functionality of the MSC

When receiving a short message TPDU from the SMS-GSMC, the MSC is responsible for the following operations:

- reception of the short message TPDU;
- retrieving information from the VLR(sendInfoForMT-SMS); location area address and, when appropriate, error information

if errors are indicated by the VLR:
- returning the appropriate error information to SMS-GSMC failure report (negative outcome of forwardShortMessage),

if no errors are indicated by the VLR:
- transferring the short message to the MS.
When receiving a confirmation that the message is received by the MS:
- relaying the delivery confirmation to the SMS-GMSC in a delivery report

When receiving a failure report of the short message transfer to the MS
- returning the appropriate error information to the SMS-GMSC in a failure report.

When receiving a notification from the MS that it has memory available to receive
one or more short messages:
- relaying the notification to the VLR

If errors are indicated by the VLR:
- returning the appropriate error information to the SMS-GMSC in a failure report.

3.6.2 MSC Functionality Related to SM MO

3.6.2.1 Functionality of the MSC

When receiving a short message TPDU from the MS, the MSC is responsible
for the following operations:

- reception of the short message TPDU
- retrieving information from the VLR. The retrieval of information from the VLR is
followed by the VLR investigating the MWF.

If errors are indicated by the VLR:
- returning the appropriate error information to the MS in a failure report (negative
outcome of sendInfoFromMO-SMS)

If no errors are indicated by the VLR:
- inspection of the PDU parameters.

If the parameters are incorrect:
- returning the appropriate error information to the MS in a failure report.

If no parameter errors are found
- examination of the destination address;
- transferring the short message TPDU to the SMS-IWMSC

When receiving the report of the short message from the SMS-IWMSC, the MSC is
responsible for the following operations:
- relaying the report to the MS.

3.6.2.2 Functionality of the SMS-IWMSC

When receiving a short message TPDU from the MSC, the SMS-IWMSC is
responsible for the following operations:
- reception of the short message TPDU;
- establishing, where necessary, a link with the addressed SC;
- transferring the short message TPDU to the SC (if the address is valid)

If a report associated with the short message is received from the SC, the SMS-IWMSC is responsible for the following operations:
- relaying of the report to the MSC

If a report associated with the short message is received from the SC before a timer expires or if the SC address is invalid, the SMS-IWMSC is responsible for the following operations:
- returning the appropriate error information to the MSC in a failure report
The value of the timer is dependent on the protocol between the SC and the SMS-IWMSC.

3.6.2.3 SMS-IWMSC functionality related to alerting

When receiving an alert from the HLR, the SMS-IWMSC is responsible for the following operations:
- inspect the SC address;
- generate an RP-Alert-SC
- transferring the RP-Alert-SC to the SC.

3.7 PDU TYPE MESSAGES AT SM-TL

There are six PDU's.

SMS-DELIVER, conveying a short message from the SC to the MS
SMS-DELIVER-REPORT, conveying a failure cause (if necessary)
SMS-SUBMIT, conveying a short message from the MS to the SC
SMS-SUBMIT-REPORT, conveying failure cause (if necessary)
SMS-STATUS-REPORT, conveying status report from the SC to the MS
SMS-COMMAND, conveying a command from the MS to the SC.

3.7.1 SMS-DELIVER Type

Basic elements of the SMS-DELIVER type:

<table>
<thead>
<tr>
<th>TP-MTI</th>
<th>TP-Message-Type-Indicator 2b</th>
<th>Parameter describing the message type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP-MMS</td>
<td>TP-More-Messages-to-Send b</td>
<td>Parameter indicating whether or not there are more messages to send</td>
</tr>
<tr>
<td>TP-RP</td>
<td>TP-Reply-Path 1octet</td>
<td>Parameter indicating the request for Reply Path</td>
</tr>
<tr>
<td>TP-UDHI</td>
<td>TP-User-Data-Header-Indicator b</td>
<td>Parameter indicating that the TP-UD field contains a Header</td>
</tr>
<tr>
<td>TP-SRI</td>
<td>TP-Status-Report-Indication b</td>
<td>Parameter indicating if the SME has requested a status report.</td>
</tr>
<tr>
<td>TP-OA</td>
<td>TP-Originating-Address 2-12o</td>
<td>Address of the originating SME.</td>
</tr>
<tr>
<td>TP-PID</td>
<td>TP-Protocol-Identifier</td>
<td>0</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------</td>
<td>---</td>
</tr>
<tr>
<td>TP-DCS</td>
<td>TP-Data-Coding-Scheme</td>
<td>70</td>
</tr>
<tr>
<td>TP-SCTS</td>
<td>TP-Service-Centre-Time-Stamp</td>
<td>70</td>
</tr>
<tr>
<td>TP-UDL</td>
<td>TP-User-Data-Length</td>
<td>1</td>
</tr>
<tr>
<td>TP-UD</td>
<td>TP-User-Data</td>
<td></td>
</tr>
</tbody>
</table>

### 3.7.2 SMS-DELIVER-REPORT Type

Basic elements of the SMS-DELIVER-REPORT type:

<table>
<thead>
<tr>
<th>TP-MTI</th>
<th>TP-Message-Type-Indicator</th>
<th>2bit</th>
<th>Parameter describing the message type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP-FCS</td>
<td>TP-Failure-Cause</td>
<td>1octet</td>
<td>Parameter indicating the reason for SMS-DELIVER failure.</td>
</tr>
</tbody>
</table>

The SMS-DELIVER-REPORT TPDU is carried as a RP-User-Data element within RP-ERROR PDU, and is part of a negative acknowledgment to a SMS-DELIVER or SMS-STATUS-REPORT. Bits 7 - 2 in octet are presently unused and sender shall set them to zero. If any of these bits is non-zero, the receiver shall not examine the other field and shall treat the TP-Failure-Cause as "Unspecified error cause".

### 3.7.3 SMS-SUBMIT Type

Basic elements of the SMS-SUBMIT type

<table>
<thead>
<tr>
<th>TP-MTI</th>
<th>TP-Message-Type-Indicator</th>
<th>2bit</th>
<th>Parameter describing the message type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP-RD</td>
<td>TP-Reject-Duplicates</td>
<td>1b</td>
<td>Parameter indicating whether or not the SC shall accept an SMS-SUBMIT for an SM still held in the SC which has the same TP-MR and the same TP-DA as a previously submitted SM from the same OA.</td>
</tr>
<tr>
<td>TP-VPF</td>
<td>TP-Validity-Period-Format</td>
<td>2b</td>
<td>Parameter indicating whether or not the TP-VP field is present.</td>
</tr>
<tr>
<td>TP-RP</td>
<td>TP-Reply-Path</td>
<td>1octet</td>
<td>Parameter indicating the request for Reply Path.</td>
</tr>
<tr>
<td>TP-UDHI</td>
<td>TP-User-Data-Header-Indicator</td>
<td>b</td>
<td>Parameter indicating that the TP-UD field contains a Header.</td>
</tr>
<tr>
<td>TP-SRR</td>
<td>TP-Status-Report-Request</td>
<td>b</td>
<td>Parameter indicating if the MS is</td>
</tr>
<tr>
<td>TP-MR</td>
<td>TP-Message-Reference</td>
<td>1</td>
<td>requesting a status report Parameter identifying the SMS-SUBMIT.</td>
</tr>
<tr>
<td>TP-DA</td>
<td>TP-Destination-Address</td>
<td>2-120</td>
<td>Address of the destination SME.</td>
</tr>
<tr>
<td>TP-PID</td>
<td>TP-Protocol-Identifier</td>
<td>0</td>
<td>Parameter identifying the coding scheme within the TP-User-Data</td>
</tr>
<tr>
<td>TP-DCS</td>
<td>TP-Data-Coding-Scheme</td>
<td>1</td>
<td>Parameter identifying the coding scheme within the TP-User-Data</td>
</tr>
<tr>
<td>TP-VP</td>
<td>TP-Validity-Period</td>
<td>0/70</td>
<td>Parameter identifying the time from where the message is no longer valid.</td>
</tr>
<tr>
<td>TP-UDL</td>
<td>TP-User-Data-Length</td>
<td>1</td>
<td>Parameter indicating the length of TP-User-Data field to follow.</td>
</tr>
<tr>
<td>TP-UD</td>
<td>TP-User-Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.7.4 SMS-SUBMIT-REPORT Type

Basic elements of the SMS-SUBMIT-REPORT type:

| TP-MTI  | TP-Message-Type-Indicator | 2bit | Parameter describing the message type. |
| TP-FCS  | TP-Failure-Cause | 1octet | Parameter indicating the reason for SMS-SUBMIT failure. |

The SMS-SUBMIT-REPORT TPDU is carried as a RP-User-Data element within RP-ERROR PDU, and is part of a negative acknowledgment to a SMS-SUBMIT or SMS-COMMAND. Bits 7 - 2 in octet are presently unused and sender shall set them to zero. If any of these bits is non-zero, the receiver shall not examine the other field and shall treat the TP-Failure-Cause as "Unspecified error cause".

### 3.7.5 SMS-STATUS-REPORT Type

Basic elements of the SMS-COMMAND type

| TP-MTI  | TP-Message-Type-Indicator | 2bit | Parameter describing the message type. |
| TP-MR   | TP-Message-Reference | 1 | Parameter identifying the SMS-SUBMIT. |
| TP-MMS  | TP-More-Messages-to-Send | b | Parameter indicating whether or not there are more messages to send. |
| TP-RA   | TP-Recipient-Address | 2-120 | Address of the recipient of the previously submitted mobile originated short message. |
| TP-SCTS | TP-Service-Centre-Time-Stamp | 7o | Parameter identifying time when |
TP-DT  TP-Discharge-Time  70 the SC received the message.
TP-ST  TP-Status  o Parameter identifying the status outcome.
of the previously sent mobile originated short message.

3.7.6 SMS-COMMAND Type

Basic elements of the SMS-COMMAND type

TP-MTI  TP-Message-Type-Indicator  2bit Parameter describing the type
TP-MR  TP-Message-Reference  I Parameter identifying the SMS-SUBMIT.
TP-SRR  TP-Status-Report-Request  b Parameter indicating if the MS is requesting a status report
TP-PID  TP-Protocol-Identifier  o Parameter identifying the coding scheme within the TP-User-Data.
TP-CT  TP-Command-Type  o Parameter specifying which operation is to be performed on a SM.
TP-MN  TP-Message-Number  o Parameter indicating which SM in the SC to operate on.
TP-DA  TP-Destination-Address  2-12o Parameter indicating the DA to which the TP-Command refers.
TP-CDL  TP-Command-Data-Length  I Parameter indicating the length of the TP-CD field.
TP-CD  TP-Command-Data  o Parameter containing user data.

3.8 DEFINITION OF THE TPDU PARAMETERS

The TP-Message-Type-Indicator is a 2 bit field, located within bits no 0 and 1 of the first octet of SMS-DELIVER, SMS-SUBMIT, SMS-STATUS-REPORT or SMS-COMMAND and to be given the following values:

<table>
<thead>
<tr>
<th>bit0</th>
<th>bit1</th>
<th>Message type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>SMS-DELIVER (in the direction SC to MS)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>SMS-DELIVER-REPORT (in the direction MS to SC)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>SMS-STATUS-REPORT (in the direction SC to MS)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>SMS-COMMAND (in the direction MS to SC)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>SMS-SUBMIT (in the direction MS to SC)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

All parameters are listed below:
3.8.1 TP-More-Message-to-Send (TP-MMS)

The Tp- More-Message-to-Send is a 1-bit field, located within bit no 2 of the first octet of SMS-DELIVER and SMS-STATUS-REPORT, and to be given the following values:

| Bit no 2: | 0 | More messages are waiting for the MS in this SC |
| | 1 | No more messages are waiting for the MS in this SC |

In the case of SMS-STATUS-REPORT this parameter refers to messages waiting for the mobile to which the status report is sent. The term message in this context refers to SMS-messages or status reports.

3.8.2 TP-Validity-Period-Format (TP-VPF)

The TP-Validity-Period-Format is a 2-bit field, located within bit no 3 and 4 of the first octet of SMS-SUBMIT, and to be given the following values:

<table>
<thead>
<tr>
<th>Bit3</th>
<th>Bit4</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>TP-VP field not present</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>TP-VP field present and integer represented</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>TP-VP field present and semi-octet represented</td>
</tr>
</tbody>
</table>

3.8.3 TP-Status-Report-Indication (TP-SRI)

The TP-Status-Report-Indication is a 1-bit field, located within bit no 5 of the first octet of SMS-DELIVER, and to be given following values:

<table>
<thead>
<tr>
<th>Bit no 5:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A status report will not be returned to the SME</td>
</tr>
<tr>
<td>1</td>
<td>A status report will be returned to the SME</td>
</tr>
</tbody>
</table>

3.8.4 TP-Status-Report-Request (TP-SRR)

The TP-Status-Report-Request is a 1-bit field, located within bit no 5 of the first octet of SMS-SUBMIT and SMS-COMMAND, and to be given the following values:

<table>
<thead>
<tr>
<th>Bit no 5:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A status report is not requested</td>
</tr>
<tr>
<td>1</td>
<td>A status report is requested</td>
</tr>
</tbody>
</table>

3.8.5 TP-Message-Reference (TP-MR)

The TP-Message-Reference gives an integer representation of a reference number of the SMS-SUBMIT or SMS-COMMAND submitted to the SC by the MS. The MS increments TP-Message-Reference by 1 for each SMS-SUBMIT or SMS-COMMAND being submitted. The value to be used for each SMS-SUBMIT is obtained by reading the Last-Used-TP-MR value from the SMS status data field in the SIM and incrementing this value by 1. After each SMS-SUBMIT has been
submitted to the network, the Last-Used-TP-MR value in the SIM is updated with the TP-MR that was used in the SMS-SUBMIT operation. The reference number may process values in the range 0 to 255. The value in the TP-MR assigned by the MS is the same value which is received at the SC.

In the case where no acknowledgment is received in response to an SMS-SUBMIT or SMS-COMMAND, than the MS may automatically repeat the SMS-SUBMIT or SMS-COMMAND but must use the same TP-MR value. The number of times the MS may repeat the SMS-SUBMIT or SMS-COMMAND is an implementation matter.

If all automatic attempts fail, the user shall be informed. The failed message shall be stored in the mobile in such a way that the user can request a retransmissions using the same TP-MR value, without needing to re-enter any information. Such storage need only be provided for a single failed message, the one most recently attempted.

The SC may discard an SMS-SUBMIT or SMS-COMMAND which has the same TP-MR value as the previous SMS-SUBMIT or SMS-COMMAND received from the same originating address. The SMS-STATUS-REPORT also contains a TP-Message-Reference field. The value sent to the MS will be the same as the TP-Message-Reference value generated by the MS in the earlier SMS-SUBMIT or SMS-COMMAND to which the status report relates.

3.8.6 TP-Originating-Address (TP-OA)

The TP-Originating-Address field is formatted according to the formatting rules of address fields.

3.8.7 TP-Destination-Address (TP-DA)

The TP-Destination-Address field is formatted according to the formatting rules of address fields.

3.8.8 TP-Protocol-Identifier (TP-PID)

The TP-Protocol-Identifier parameter serves the purposes indicated previous section. It consist of one octet, and the bits in the octet are used as follows: The MS will not interpret reserved or unsupported values but shall store them as received. The SC may reject messages with a TP-Protocol-Identifier containing a reserved value or one which is not supported.

<table>
<thead>
<tr>
<th>bits</th>
<th>usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>Assign bits 0..5 as defined below</td>
</tr>
<tr>
<td>0</td>
<td>Assign bits 0..5 as defined below</td>
</tr>
<tr>
<td>1</td>
<td>reserved</td>
</tr>
<tr>
<td>1</td>
<td>Assign bits 0-5 for specific use.</td>
</tr>
</tbody>
</table>
in the case where bit 7 = 0 and bit 6 = 0, bit 5 indicates telematic interworking:
value = 0 : no interworking, but SME-to-SME protocol
value = 1 : telematic interworking

In the case of telematic interworking, the following five bit patterns in bits 4..0 are used to indicate different types of telematic devices.
Note that for the straightforward case of simple MS-to-SC short message transfer the TP-Protocol-Identifier is set to value 0.

3.8.9 TP-Data-Coding-Scheme (TP-DCS)

The TP-Data-Coding-Scheme indicates the data coding scheme of the TP-UD field, and may indicate message class.

3.8.10 TP-Service-Centre-Time-Stamp (TP-SCTS)

The TP-Service-Centre-Time-Stamp field is given in semi-octet representation, and represents the local time in the following way:

<table>
<thead>
<tr>
<th>Digits:</th>
<th>Year:</th>
<th>Month:</th>
<th>Day:</th>
<th>Hour:</th>
<th>Minute:</th>
<th>Second:</th>
<th>TimeZone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

The Time Zone indicates the difference, expressed in quarters of an hour, between the local time and GMT. In the first of the two semi-octets, the first bit represents the algebraic sign of this difference (0: positive, 1: negative).
The TP-Service-Centre-Time-Stamp, and any other times coded in this format, represents the time local to the sending entity. The time zone code enables the receiver to calculate the equivalent time in GMT from the other semi-octets in the TP-Service-Centre-Time-Stamp, or indicate the time zone (GMT, GMT+1H etc.), or perform other similar calculations as required by the implementation.

3.8.11 TP-Validity-Period

The TP-Validity-Period field is given in either integer or semi-octet representation. In the first case, the TP-Validity-Period comprises 1 octet, giving the length of the validity period, counted from when the SMS-SUBMIT is received by the SC. In the second case, the TP-Validity-Period comprises 7 octets, giving the absolute time of the validity period termination.

In the first case, the representation of time is as follows:

<table>
<thead>
<tr>
<th>TP-VP value</th>
<th>Validity period value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 143</td>
<td>(TP-VP + 1) * 5 minutes</td>
</tr>
<tr>
<td>144 to 167</td>
<td>12 hours + ( (TP-VP - 143) * 30 minutes )</td>
</tr>
<tr>
<td>168 to 196</td>
<td>(TP-VP - 166) * 1 day</td>
</tr>
<tr>
<td>197 to 255</td>
<td>(TP-VP - 192) * 1 week</td>
</tr>
</tbody>
</table>

In the second case, the representation of time is identical to the representation of the TP-Service-Centre-Time-Stamp.
3.8.12 TP-Discharge-Time (TP-DT)

The TP-Discharge-Time field indicates the time at which a previously submitted SSM-SUBMIT was successfully delivered to or attempted to deliver to the recipient SME or disposed of by the SC. In the case of transaction completed the time shall be the time of the completion of the transaction. In the case of "SC still trying to transfer SM" the time shall be the time of the last transfer attempt. In the case of "permanent or temporary error - SC not making any more transfer attempts" the time shall be the time of either the last transfer attempt or the time at which the SC disposed of the SM according to the Status outcome in TP-ST. The TP-Discharge-Time is given in semi-octet representation in a format identical to the TP-SCTS.

3.8.13 TP-Recipient-Address (TP-RA)

The TP-Recipient-Address field indicates the address of the SME that was the destination of the previously submitted mobile originated short message being subject to the status report. The field is formatted according to the formatting rules of address fields.

3.8.14 TP-Status (TP-ST)

The TP-Status field indicates the status of a previously submitted SMS-SUBMIT and certain SMS-COMMANDS for which a Status-Report has been requested. It consist of one octet and the bits in the octet are used as follows:

The MS will not interpret any reserved values but shall store them as received.

<table>
<thead>
<tr>
<th>bits</th>
<th>value/usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>6..0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short message transaction completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
</tr>
<tr>
<td>0000001</td>
</tr>
<tr>
<td>0000010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reserved values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000011 .. 0001111</td>
</tr>
<tr>
<td>0010000 .. 0011111</td>
</tr>
</tbody>
</table>

Temporary error are listed ,SC still trying to transfer SM.

| 0100000 | Congestion |
0100001  SME busy
0100010  No response from SME
0100011  Service rejected
0100100  Quality of service not available
0100101  Error in SME
0100110..0101111  Reserved
0110000..0111111  Values specific to each SC

Permanent error are listed, SC is not making any more transfer attempts

1000000  Remote procedure error
1000001  Incompatible destination
1000010  Connection rejected by SME
1000011  Not obtainable
1000100  Quality of service not available
1000101  No interworking available
1000110  SM validity period expired
1000111  SM deleted by originated SME
1001000  SM deleted by SC administration
1001001  SM does not exist
1001010..1001111  reserved
1010000..1011111  Values specific to each SC

Temporary error, SC is not making any more transfer attempts

1100000  Congestion
1100001  SME busy
1100010  No response from SME
1100011  Service rejected
1100100  Quality of service not available
1100101  Error in SME
1100110..1101111  Reserved
1110000..1111111  Values specific to each SC

If bits 7 is set to 1, the following bits are reserved for manufacturer.

3.8.15 TP-User-Data-Length (TP-UDL)

The TP-User-Data is coded using the default alphabet, the TP-User-Data-Length field gives an integer representation of the number of characters within the TP-User-Data field to follow. If the TP-User-Data-Header field is included in the TP-User-Data field then the TP-User-Data-Length field still gives an integer representation of the number of septets in the entire TP-User-Data field and includes the TP-User-Data-Header field in its count.
If the Tp-User-Data is coded using 8-bit data, the TP-User-Data-Length field gives an integer representation of the number of octets within the TP-User-Data field to follow.

3.8.16 TP-Reply-Path (TP-RP)

The TP-Reply-Path is a 1-bit field, located within bit no 7 of the first octet of both SMS-DELIVER and SMS-SUBMIT, and to be given the following values:

Bit no 7: 0 TP-Reply-Path parameter is not sent in this SMS-SUBMIT/DELIVER
        1 TP-Reply-Path parameter is sent in this SMS-SUBMIT/DELIVER

3.8.17 TP-Message-Number (TP-MN)

The TP-Message-Number is an 8-bit field allowing an MS to refer uniquely to an SM in the SC which that MS has previously submitted. The TP-MN value is the TP-MR value of a previously submitted SM.

3.8.18 TP-Command-Type (TP-CT)

The TP-Command-Type is an 8-bit field specifying the type of operation that the SC is to perform. It has the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Command description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>Enquiry relating to previously submitted short message</td>
</tr>
<tr>
<td>00000001</td>
<td>Cancel Status Report Request relating to previously submitted short message</td>
</tr>
<tr>
<td>00000010</td>
<td>Delete previously submitted Short Message</td>
</tr>
<tr>
<td>00000011</td>
<td>Enable Status Report Request relating to previously submitted short message</td>
</tr>
<tr>
<td>00001000..0001111</td>
<td>Reserved</td>
</tr>
<tr>
<td>11100000..1111111</td>
<td>Values specific for each SC</td>
</tr>
</tbody>
</table>

The SC will return an RP-Error with an appropriate TP-Failure-Cause for any TP-Command value which is reserved, unsupported or invalid or the actioning of the command has failed.

The SC will return an RP-ACK if the actioning of the Command has succeeded. A successful Enquiry will result in the SC sending a SMS-STATUS-REPORT for the SM to which the Enquiry refers. In the case where the SC has a number of SMSs which have the same TP-MR, the same TP-DA and have come from the same originating address the SC will send a SMS-STATUS-REPORT for each SM.

In the case where a TP-Command is to Delete a previously submitted short message, the SC will send a Status Report indicating that the SM has been deleted if the original Submit SM requested a status report.
3.8.19 TP-Command-Data-Length (TP-CDL)

The TP-Command-Data-Length field is used to indicate the number of octets contained within the TP-Command-Data-field. If this field is set to zero, the TP-Command-Data field will not be present.

3.8.20 TP-Command-Data (TP-CD)

The TP-Command-Data field contains data relating to the operation requested by the MS which is to be performed at the SC. The maximum length of this field is 157 octets. The usage and provision of the optional TP-Command-Data field will be determined by the function selected by the TP-Command-Type field.

3.8.21 TP-Failure-Cause (TP-FCS)

The TP-Failure-Cause field is used to report the reason for failure to transfer or process a short message.

3.8.22 TP-User-Data-Header-Indicator (TP-UDHI)

The TP-User-Data-Header-Indicator is a 1 bit field within bit 6 of the first octet of an SMS-SUBMIT and deliver SMS-DELIVER PDU and has the following values.

<table>
<thead>
<tr>
<th>Bit no</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

3.8.23 TP-User Data (TP-UD)

The TP-User-Data field contains up to 140 octets of user data. The TP-User-Data field may comprise just the short message itself or a Header in addition to the short message depending upon the setting of TP-UDHI. Where the TP-UDHI value is set to 0 the TP-User-Data field comprises the short message only. Where the TP-UDHI value is set to 1 the first octets of the TP-User-Data field contains a Header in the following order starting at the least significant octet of the TP-User-Data field.

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of the User Data Header</td>
<td>1 octet</td>
</tr>
<tr>
<td>Information-Element-Identifier 'A'</td>
<td>1 octet</td>
</tr>
<tr>
<td>Length of Information-Element 'A' Data</td>
<td>1 octet</td>
</tr>
<tr>
<td>Information Element 'A' Data</td>
<td>1 ..n octets</td>
</tr>
<tr>
<td>Information-Element-Identifier 'B'</td>
<td>1 octet</td>
</tr>
<tr>
<td>Length of Information-Element 'B' Data</td>
<td>1 octet</td>
</tr>
<tr>
<td>Information Element 'B' Data</td>
<td>1 ..n octets</td>
</tr>
</tbody>
</table>
SECTION 4

ANALYSIS OF GSM 07.07 AT COMMANDS

This chapter contains a detailed list of all commands specified in GSM 07.07. Each command is described by its name and associated actions. Furthermore, each command the functional group to which the commands belongs is given.

4.1 TE - TA INTERFACE COMMANDS

All of the parameters which are updated and listed below must be stored in non-volatile memory. If there is a &W in somewhere of the commands then these parameters must be stored in non-volatile memory.

4.1.1 Write Command Line Termination

Command : S3=[<value>]
Action: Command to be handled in TA by setting the command line termination character to <value> and handle all the commands and responses ending with that <value>.

4.1.2 Write Response Formatting Character

Command : S4=[<value>]
Action: Command to be handled in TA by setting the response formatting character to <value> and handle all the commands and responses ending with that <value>.

4.1.3 Write Command Line Editing Character

Command : S5=[<value>]
Action: Command to be handled in TA by setting command line editing character to <value>
4.1.4 Write Echo Mode

Command: E[<value>]
Action: Command to be handled in TA by setting echo mode according to the <value>.
If <value> = 0 TA does not echo the commands.
If <value> = 1 TA echoes the commands.

4.1.5 Write Result Code Suppression Mode

Command: Q[<value>]
Action: Command to be handled in TA by setting result code suppression mode according to the <value>.
If <value> = 0 TA transmits result codes.
If <value> = 1 TA does not transmit result codes.

4.1.6 Write Result Code Format

Command: V[<value>]
Action: Command to be handled in TA by setting response format mode according to the <value>.
If <value> = 0 TA transmits result codes in numeric format.
If <value> = 1 TA transmits result codes in verbose format.

4.1.7 Write Connect Result Code Format

Command: X[<value>]
Action: Command to be handled in TA by setting CONNECT result code format mode according to the <value>.

4.1.8 Write Data Carrier Detection Mode

Command: &C[<value>]
Action: Command to be handled in TA by setting Data Carrier Detection Mode to <value>.
If <value> = 0 TA does not care the DCD signal.
If <value> = 1 TA operation relates to detection of DCD signal.
4.1.9 Write DTR Mode

Command:  &D[<value>]
Action: Command to be handled in TA by setting DTR mode according to the <value>.
If <value> = 0 TA cuts the connection if DTR changed OFF.
If <value> = 1 TA does not care the DTR signal changes.

4.2 GENERAL COMMANDS

4.2.1 Read Manufacturer Information of ME

Command:  +CGMI
Response of TA:  <manufacturer>
+CME ERROR: <err>
Action: get the information about the manufacturer data of the ME and sends it to TE.

4.2.2 Read Model Information of ME

Command:  +CGMM
Response of TA:  <model>
+CME ERROR: <err>
Action: get the information about the model of the ME and send it to TE.

4.2.3 Read Version Information of ME

Command:  +CGMR
Response of TA:  <revision>
+CME ERROR: <err>
Action: get the information about the version, revision date or other information of the ME and send it to TE.

4.2.4 Read IMEI of ME

Command:  +CGSN
Response of TA:  <sn>
+CME ERROR: <err>
Action: get the information about the of the ME and send it to TE.
4.2.5 Write Character Set

Command: +CSCS=<chset>
Action: Command to be handled in TA by setting character set to <chset>
Possible values of <chset> "GSM", "HEX", "IRA", "PCCP", "8859-1", "PCDN".

4.2.6 Read Current Character Set

Command: +CSCS?
Response of TA: +CSCS: <chset>
Action: Command to be handled in TA by sending the current character set in use.

4.2.7 Read Supported Character Sets

Command: +CSCS=?
Response of TA: +CSCS: <list of supported <chset>s>
Action: Command to be handled in TA by sending the supported character sets.

4.2.8 Write to Default

Command: &F[<value>]
Action: Command to be handled in TA by setting all parameters to the default specified by <value>.

4.2.9 Read Complete Capabilities

Command: +GCAP
Response of TA: +GCAP: +CGSM, +FCLASS, +W
Action: Command to be handled in TA by sending fax, data capabilities of TA.

4.2.10 Write Country Code

Command: +GCI=<T.35>
Action: Command to be handled in TA by setting the country code.
4.3 CALL CONTROL COMMANDS

4.3.1 Write Dial Number Type

Command: +CSTA=[<type>]
Action: Command to be handled in TA by setting the type of number for dialling. Possible values of <type>
145: when dialling string includes international access code character “+”.
129: when dialling string doesn’t include international access code character “+”.

4.3.2 Read Current Dial Number Type

Command: +CSTA?
Response of TA: +CSTA: <type>
Action: Command to be handled in TA by sending the type of number for further dialling command.

4.3.3 Read Supported Dial Number Types

Command: +CSTA=?
Response of TA: +CSTA: (list of supported <type>s)
Action: Command to be handled in TA by sending the supported types of number for further dialling commands.

4.3.4 Write Call Mode

Command: +CMOD=[<mode>]
Response of TA: +CME ERROR: <err>
Action: set the call mode to <mode>.
Possible values of <mode>
0: Single mode
1: Alternate mode

4.3.5 Read Current Call Mode

Command: +CMOD?
Response of TA: +CMOD: <mode>
Action: Get the call mode <mode> through MN and send it to TE.
4.3.6 Read Supported Call Modes

Command: CMOD=?
Response of TA: +CMOD: (list of supported <mode>s)
Action: Get the supported list of call modes <mode>s and send it to TE.

4.3.7 Terminate Current Call

Command: +CHUP
Action: Terminates the current GSM call.

4.3.8 Read Result of Last Unsuccessful Call

Command: +CEER?
Response of TA: +CEER: <report>
Action: Gets the result code of last unsuccessful call and converts this code to a single line <report> and sends it to TE.

4.3.9 Send Setup Response

Command: A
Action: Sends a setup response.

4.3.10 Terminate Current Call

Command: H
Action: Terminates current GSM call through MN.

4.3.11 Switch to Data Mode

Command: O[<value>]
Action: Command to be handled in TA by changing its mode from command to data.

4.3.12 Write Answer Mode

Command: S0=[<value>]
Action: Command to be handled in TA by setting S0 parameter to <value> so as to handle the incoming calls.
Possible values of <value>
0: Manual answering
1..XX Automatic answering

4.3.13 Write Call Terminate Request Timeout

Command:  S7=[<value>]
Action: Command to be handled in TA by setting S7 parameter to <value>
so as to wait for completion of call answering or originating procedure before sending a
 call terminate request.

4.3.14 Write Digit Send Timeout

Command:  S8=[<value>]
Action: Command to be handled in TA by setting S8 parameter to <value>
seconds so as to wait.

4.3.15 Write Call Termination Timeout

Command:  S10=[<value>]
Action: Command to be handled in TA by setting S10 parameter to <value>
tenths of seconds to wait for terminating the call after the indication of the absence of
received line signal.

4.4 NETWORK SERVICE RELATED COMMANDS

4.4.1 Write Network Registration Presentation Status

Command:  +CREG=[<value>]
Action: Command to be handled in TA by setting the status of result code
 presentation.

Possible values of <value>
0 Disable network registration unsolicited result code.
1 Enable network registration unsolicited result code.

4.4.2 Read Current Network Registration Status

Command:  +CREG?
Action: Command to be handled in TA by getting the network change
information and sends it to TE.

Possible values of <stat>
0 not registered, ME is not currently searching a new operator to register to.
1 registered
2 not registered, but ME is currently searching a new operator to register to.
3 registration denied.
4 unknown

4.4.3 Read Supported Network Registration Presentation Status

Command:  CREG=?
Action: Command to be handled in TA by sending the possible result code presentations <n>s to TE.

4.4.4 Register GSM Network Operator

Command:  COPS=[<mode>[,<format>[,<oper>]]]
Action: Selects and registers the GSM network operator <mode>.
Possible values of <mode>
0 automatic (<oper> field is ignored)
1 manual (oper field shall be present)
2 deregister from network
3 set only <format> (for read command +COPS), do not attempt registration/deregistration (<oper> field is ignored)

Possible values of <format>
0 long format alphanumeric <oper>
1 short format alphanumeric <oper>
2 numeric <oper>
   <oper> string type given by <format>

Possible values of <stat>
0 unknown
1 available
2 current
3 forbidden

4.4.5 Read Registered GSM Network Operator

Command:  +COPS?
Response of TA:  +COPS:<model>[,<format>,<oper>]
+CME ERROR: <err>
Action: Gets the current mode and the currently selected operator sends it to TE. If no operator is selected <format> and <oper> are omitted.

4.4.6 Read Present GSM Network Operators

Command: +COPS=?
Response of TA: +COPS: list of supported (<stat>, long alphanumeric <oper>, short alphanumeric <oper>, numeric <oper>s
+CME ERROR: <err>
Action: Gets the present operator characteristics and sends it to TE.

4.4.7 Write CLIP Status

Command: +CLIP=<n>
Response of TA: +CME ERROR: <err>
Action: Sets the status of presentation of CLI in TA.

4.4.8 Read CLIP Status

Command: CLIP?
Response of TA: +CLIP: <n>,<m>
Action: Gets subscriber CLIP service status sends it to TE with the CLIP status in MT.
Possible values of <m>
0 CLIP not provisioned
1 CLIP provisioned
2 unknown (e.g. no network)

4.4.9 Read Supported CLIP Status

Command: +CLIP=?
Response of TA: +CLIP(list of supported <n>s)
Action: Send the status presentation values of CLI to TE.
4.4.10 Write CLIR Status

Command:  +CLIR=<n>
Response of TA:  +CME ERROR:<err>
Action:  Sets the status of presentation of CLI to the called party.
Possible values of <n> (sets the temporary mode status in the TA)

0  set the presentation indicator according to the subscription of the CLIR service.
1  override the default value of the presentation indicator.
2  CLIR invocation
3  CLIR suppression.

4.4.11 Read CLIR Status

Command:  +CLIR?
Response of TA:  +CLIR:<d>,<m>
Action:  Gets subscriber CLIR service status and sends it to TE with the temporary mode status in MT.
Possible values of <d>: shows the temporary mode status in MT.
0  CLIR temporary mode presentation restricted.
1  CLIR temporary mode presentation allowed.
Possible values of <m>: shows the subscriber CLIR service status in the network
0  CLIR not provisioned
1  CLIR provisioned in permanent mode
2  unknown (e.g. no network)
3  CLIR temporary mode presentation restricted
4  CLIR temporary mode presentation allowed

4.4.12 Read Supported CLIR Status

Command:  CLIR=?
Response of TA:  +CLIR: (list of supported <n>s)
Action:  Sends the status presentation values of CLI to the called party to TE.
4.5 MOBILE EQUIPMENT CONTROL AND STATUS COMMANDS

4.5.1 Read ME Status

Command:  +CPAS
Response of TA:  +CPAS: <pas>
                  +CME ERROR: <err>
Action:  Reads the information about the TE activity status and sends it to TE.

Possible values of <pas>
0   ready (ME allows commands from TA/TE)
1   unavailable (ME does not allow commands from TA/TE)
2   unknown (ME is not guaranteed to respond the instructions).
3   ringing (ME is ready for commands from TA/TE but the ringer is active)
4   call in progress (ME is ready for commands from TA/TE but a call is in progress)
5   asleep (ME is unable to process commands from TA/TE because it is in a low functionality state.)

4.5.2 Read Supported ME Status

Command:  +CPAS=-?
Response of TA:  +CPAS: (list of supported <pas>s)
                  +CME ERROR: <err>
Action:  Gets the supported activity status and sends it to TE.

4.5.3 Write New Pin

+CPIN=<pin>[,<newpin>]
Response of TA:  +CME ERROR: <err>
Action:  Sends a password <pin> if <newpin> is also given the new password is set to <newpin>.

4.5.4 Read Waiting Password Type

Command:  +CPIN?
Response of TA:  +CPIN:<code>
                  +CME ERROR: <err>
Action: Reads from DM which password ME is waiting for.

Possible values of <code>
READY ME is not pending for any password.
SIM PIN ME is waiting SIM PIN to be given.
SIM PUK ME is waiting SIM PUK to be given.
PH-SIM PIN ME is waiting Phone - to - SIM card password to be given.

4.5.5 Read Battery Status

Command: +CBC
Response of TA: +CBC:
Action: Gets battery connection status and battery charge level sends it to TE.

Possible values of <bcs>
0 ME is powered by the battery
1 ME has a battery connected, but is not powered by it.
2 ME does not have battery connected
3 Recognized power fault, calls inhibited possible values of <bcl>

Possible values of <bcl>
0 battery is exhausted, or ME does not have a battery connected
1.. 100 battery has 1 - 100 percent of capacity remaining

4.5.6 Read Supported BCS and BCL Types

Command: +CBC=?
Response of TA: +CBC: (list of supported <bcs>s),(list of supported <bcl>s)
Action: Gets a list of battery connection status and battery charge level and sends it to TE.

4.5.7 Read Signal Level and Error Rate

Command: CSQ
Response of TA: CSQ: <rssi>,<ber>
+CME ERROR: <err>
Action: Gets received signal strength indication and channel bit error rate and sends it to TE.

Possible values of <rssi>
0 113dBm or less
1 111 dBm
2.. 30  109.. -53 dBm
31 51 dBm or greater
99 not unknown or not detectable

possible values of <ber>

0.. 7 As RXQUAL values in the table in GSM 05.08
99 not known or not detectable

4.5.8 Read Supported Signal Level and Error Rate

Command:  +CSQ
Response of TA:  +CSQ: <rssi>,<ber>
             +CME ERROR: <err>
Action: Gets supported received signal strength indication and channel bit error rate values and sends it to TE.

4.5.9 Select Phonebook Storage Type

Command:  +CPBS=<storage>
Response of TA:  +CME ERROR: <err>
Action: Sets the phonebook memory storage in DM.

Possible values of <storage>
“FD” SIM fixdialling phonebook
“LD” SIM lastdialling phonebook
“ME” ME phonebook
“MT”combined with ME and SIM phonebook
“SM” SIM phonebook
“TA” TA phonebook

4.5.10 Read Current Phonebook Storage Type

Command:  +CPBS?
Response of TA:  +CPBS: <storage>
Action: Gets the current phonebook memory storage and sends it to TE.
4.5.11 Read Supported Phonebook Storage Type

Command:  +CPBS=?
Response of TA:  +CPBS: (list of supported <storage>s)
Action:  Gets available phonebook storages and sends them to TE.

4.5.12 Read Phonebook Entries

Command:  +CPBR=<index>[,<index2>]
Response of TA:  +CPBR:<index1>,<number>,<type>,<text>[...]
<CR><LF>+CPBR: <index2>,<number>,<type>,<text>]
Action:  Gets the phonebook entries from location <index1> to location <index2> from the phonebook storage set and sends to TE.

Read Current Phonebook Characteristics

Command:  +CPBR=?
Response of TA:  +CPBR: (list of supported <index>s),<nlength>,<tlength>
+CME ERROR: <err>
Action:  Gets the location range of the current phonebook storage maximum lengths of number and text fields of entries and sends them to TE.

4.5.13 Read Phonebook Entries Alphanumerically

Command:  +CPBF=<findtext>
Response of TA:  +CPBF: <index1>,<number>,<type>,<text>[...]<CR><LF>
+CME ERROR: <err>
Action:  Gets the phonebook entries alphanumeric starting from <findtext> from the phonebook storage set and sends them to TE.

4.5.14 Read Supported Phonebook Number and Text Length

Command:  +CPBF=?
Response of TA:  +CPBF: (list of supported <index>s),<nlength>,<tlength>
+CME ERROR: <err>
Action:  Gets maximum lengths of number and text fields of phonebook entries and sends them to TE.
4.5.15 Write Phonebook Entry

Command:   +CPBW=[<index>][<number>[,<type>],[<text>]]
Response of TA:   +CME ERROR: <err>
Action:   Writes the phonebook entry in location <index> to the phonebook storage set.

4.5.16 Read Selected Phonebook Location Range

Command:   +CPBW=?
Response of TA:   +CPBW: (list of supported <index>s,<nlength>;<list of supported <type>s),<length>
                 +CME ERROR: <err>
Action:   Gets the location range of the current phonebook storage maximum lengths of number and text fields of entries and sends them to TE.

4.6 GENERAL CONFIGURATION COMMANDS

4.6.1 Select SMS Service

Command:   +CSMS=<service>
Action:   Send the information about the selected SMS service and get the response.

<service>
0   GSM 03.40 and 03.41
1..127 reserved
<mt>,<mo>,<bm>
0   type not supported
1   type supported

4.6.2 Read Current SMS Service

Command:   CSMS?
Action:   Get the information about the current selected SMS service and send to the TE.
4.6.3 Read Supported SMS Services

Command: CSMS=?
Action: Get information about the list of supported SMS service and send to the TE.

4.6.4 Read and Write SMS Storage

Command: +CPMS=<mem1>[,<mem2>]
Action: Gets the available SMS storage and sends them to TE.
This command selects memory storages <mem1> and <mem2> to be used for reading, writing, etc.

4.6.5 Read SMS Memory Information

Command: CPMS?
Action: Get the information the all memory used from SMS service and send it to the TE.

4.6.6 Read Supported Memory List

Command: +CPMS=?
Action: Get the information about the list of supported memory.

4.6.7 Write SMS I/O Format

Command: +CMGF=[<mode>]
Action: Set the input output format of the message to use. Command to be handled in TA.
Set command tells the TA, which input and output format of messages to use.
<mode> indicates the format of messages used with send, list, read, and write commands and unsolicited result codes resulting from received messages. Mode can be either PDU mode or text mode (headers and body of the messages given as separate parameters). Text mode uses the value of parameter <chset> specified by command select TE character set +CSCS to inform the character set to be used in the message body in the TA-TE interface.

<mode>:
0 PDU mode
1 TEXT mode.
4.6.8 Read Current SMS I/O Format

Command: CMGF?
Action: Send the current input output format to the TE.

4.6.9 Read Supported SMS I/O Formats

Command: CMGF=?
Action: Send the supported input output format to the TE.

examples:

AT+CSMS=? (inquiry of available services in TA)
+CSMS: (0) (only GSM SMS implemented)
OK
AT+CSMS=0;+CPMS=? (set GSM SMS; query available memories)
+CSMS: 1,1,1 (all MT, MO and CBM supported)
+CPMS: ("BM","ME","SM"),("ME","SM") (CBM, ME and SIM memories)
OK
AT+CPMS="ME","ME";+CMGF=? (for writing)
+CPMS: "ME",5,99,"ME",5,99 (five messages in ME total 99 space)
+CMGF: (0,1) (both text and PDU mode implemented)
OK
AT+CMGF=1;+CSCS=?
+CSCS: ("IRA","PCCP437")
OK
AT+CSCS="PCCP437" (select PC code page 437)
OK

4.6.10 Write SMS Center Address

Command: +CSCA=<sca>[,<tosca>]
Action: Set the SMS Center address.

Set command updates the SMSC address, through which mobile originated SMs are transmitted. In text mode, setting is used by send and write commands. In PDU mode, setting is used by the same command.
4.6.11 Read SMS Center Address

Command: CSCA?
Action: Read the SMS CENTER address information and send it to TE.

4.6.12 Write SMS Additional Parameters

Command: +CSMP=[<fo>[,<vp>[,<pid>[,<dcs>]]]]
Action: Set the additional parameters needed when SM is sent to the network or placed in a storage.

Set command is used to select values for additional parameters needed when SM is sent to the network or placed in a storage when text format message mode is selected.

<fo>: depending on the command or result code; first octet of SMS-DELIVER, SMS-STATUS-REPORT or SMS-COMMAND in integer format.

<vp>: depending on SMS-SUBMIT <fo> setting. TP - Validity - Period either in integer format or in time string format.

<pid>: TP-Protocol-Identifier in integer format.

<dcs>: depending on the command or result code SMS Data Coding Scheme or Cell Broadcast Data Coding Scheme in integer format.

4.6.13 Read SMS Additional Parameters

Command: +CSMP?
Action: Show the current additional parameters of SM.

4.6.14 Read Supported SMS Additional Parameters

Command: +CSMP=?
Action: show the supported additional parameters.

4.6.15 Write Result Code Header Presentation Parameter

Command: +CSDH=[<show>]
Action: Control whether detailed header information is shown in text mode result codes.

<show>:
0 do not show header values defined in commands
+CSCA and +CSMP (<sca>,<tosca>,<foo>,<vp>,<pid> and <dcs>) nor <length>,<toda> in +CMT,+CMGL,+CMGR result codes for SMSPP text mode.

1 show the values in resulkt codes.

4.6.16 Read Result Code Header Presentation Parameter

Command: +CSDH?
Action: Send the <show> variable context to the ME.

4.6.17 Read Supported Result Code Header Presentation Parameter

Command: +CSDH=?
Action: Send list of supported <show>s to the ME.

4.6.18 Write CBM Message Type

Command: +CSCB=[<mode>[<mids>[<dcss>]]]
Action: Select which types of CBM are to be received by the ME.

<mode>:
0 message types specified in <mids> and <dcss> are accepted.
1 they are not accepted.

<mids>:
string type: all different possible combinations of CBM message identifiers.

<dcss>:
string type: all different possible combinations of CBM data coding schemes e.g "0-3,5"

4.6.19 Read Acceptable CBM Message Types

Command: +CSCB?
Action: Shows which types of CBMs are to be received by the ME.

4.6.20 Read Supported CBM Message Types

Command: +CSCB=?
Action: Returns the list of supported <mode>s.
4.6.21 Write Active SMS Settings

Command: +CSAS[=<profile>]
Action: Saves the active message service setting to a non-volatile memory.

Execution command saves active message settings to a non-volatile memory. A TA can contain several profiles of settings. All settings specified in commands +CSCA, +CSMP, and +CSCB are saved.

<profile>:
0.. 255 manufacturer specific profile number.

4.6.22 Read Supported SMS Setting Profile Number

Command: CSAS=?
Action: Display the supported profile number for reading and writing of settings.
examples:
+CSMP is used to set the text mode header values of SMS-SUBMIT (or SMS-DELIVER when received message is written from TE to a storage). The volatile memory may as well be in the ME, or when no volatile memory is used, +CSMP, +CSCA and +CSCB settings are stored directly to non-volatile memory of ME. In this example, the volatile parameter settings of TA are used to construct messages in text mode. SMSC address setting is used also in PDU mode. The next example illustrates a session to restore the message parameters from the ME to the TA, and to set up the CBM identifiers which are wanted to be received:

AT+CRES (restore settings from non-volatile memory to non-volatile memory)
AT+CSMP?;+CSCA? (query SM parameter)
+CSMP: 17,167,0,0 (default values for SMS-SUBMIT)
+CSCA: "+905329010000",145 (SMSC address)
OK
AT+CSDH=1 (show all headers in text mode)
OK
AT+CSCB=1 (all CBMs are accepted)
OK
4.6.23 Restore SMS Settings

Command: +CRES[<profile>]
Action: Restores message service settings from non-volatile memory to active memory. All settings specified in commands Service Centre Address +CSCA, set message parameters +CSMP and select cell broadcast message types +CSCB are restored.

4.6.24 Read Supported Profiles

Command: +CRES=?
Action: Displays the list of supported profiles.

4.7 MESSAGE RECEIVING AND READING COMMANDS

In the present application on handy, the user working through the local MMI can receive, read, store and delete the Short Messages, put members included in a short message into the phonebook, edit short messages for sending them to another user and finally send short messages to another user. The application on PC will support all of the above features, and it will be also more comfortable than the using of handy’s keypad and display. It is possible to store messages on PC, import and export databases and use it as an e-mail tool.

The CBS (Cell Broadcast Services) service is analogous to the teletext service offered on Television, in that like Teletext, it permits a number of unacknowledged general messages to be broadcast to all receivers within a particular region. To permit mobiles to selectively display only those messages required by the MS user, CBS messages are assigned a message class which categorises the type of information that they contain and the language in which the message has been composition provider. The frequency at which messages are repeatedly transmitted will be dependent on the information that they contain; for example it is likely that dynamic information such as road traffic information, will require more frequent transmission than weather information. It is more comfortable to read these messages from the PC display, instead of the MT's display. The application related with road and weather information for car can be realized with CBM service.

4.7.1 Write Message Receive Indication Procedure

Command: +CNMI=[<mode[,<mt>[.,<bm>[.,<ds>[.,bfr>]]]]]
Action: Selects the procedure how receiving of new messages from the network is indicated to the TE when TE is active.
4.7.2 Read Current Message Receive Indication Procedure

Command: +CNMI?
Response of TA: +CNMI:<mode>,<mt>,<bm>,<ds>,<bfr>
Action: Sends the current settings in TA to TE about the procedure how receiving of new messages from the network is indicated to the TE when TE is active.

4.7.3 Read Supported Message Receive Indication Procedure

Command: +CNMI=?
Response of TA: +CNMI: (list of supported <mode>s),(list of supported <mt>s),(list of supported <bm>s),(list of supported <ds>s),(list of supported <bfr>s).
Action: Sends the list of settings in TA about the procedure how receiving of new messages from the network is indicated to the TE when TE is active.

4.7.4 Read Messages with Status

Command: +CMGL=[<stat>]
Response of TA: SM storage

+CMGL:<index>,<stat>,<oa/da>[<alpha>][,<scts>][,<tooa>/<toda>,<length>
<CR><LF>
<data> <CR><LF>
...
Action: Reads messages with status <stat> and sends them to the TE.

4.7.5 Read Message

Command: +CMGR=<index>
Response of TA: SMS - DELIVER
+CMGR:<stat>,<oa>[<alpha>][,<scts>][,<tooa>,<fo>,<pid>,<dcs>[,<sca>],<tsca>,<length>][<CR><LF><data>

SMS - SUBMIT
+CMGR:<stat>,<da>[<alpha>[,<toda>,<fo>,<pid>,<dcs>[,<vp>],[<sca>],<tosca>,<length>][<CR><LF><data>
CBM storage
+CMGR:<stat>,<sn>,<mid>,<dcs>,<page>,<pages><CR><LF><data>
       Action: Reads message in location <index> from preffered message storage
       and sends it to the TE.

4.8 MESSAGE SENDING AND WRITING COMMANDS

4.8.1 Send Message

Command: +CMGS=<da>[,<toda>]<CR>text is entered<ctrl-Z/ESC>
Response of TA: if sending succesfull
       +CMGS:<mr>
       else
       +CMS ERROR: <err>
       Action: Sends message which has come from a TE to the network and
       returns a message reference <mr> to TE.

4.8.2 Send Message From Memory

Command: +CMSS=<index>[,<da>[,<toda>]]
Response of TA: if sending succesfull
       +CMSS: <mr>
       else
       +CMS ERROR: <err>
       Action: Sends message to the network with location <index> from preferred
       message storage, returns a message reference <mr> to TE.

4.8.3 Write Message

Command: +CMGW=<oa/da>[<tooa/toda>[,<stat>]]<CR>text is entered<ctrl-
       Z/ESC>
Response of TA: +CMGW:<index>
       +CMS ERROR: <err>
       Action: Stores message in preferred memory and returns the location of
       message <index> to TE.
4.8.4 Delete Message From Memory

Command:  +CMGD=<index>
Response of TA:  +CMS ERROR: <err>
Action: Deletes message in location <index> from preferred memory storage.

4.8.5 Send Command Message

Command:  +CMGC=<fo>,<ct>[,<pid>[,<mn>[,<da>[,<toda>]]]]<CR>text is entered<ctrl-Z/ESC>
Response of TA:  if successful
+CMGC: <mr>
else
+CMS ERROR: <err>
Action: Sends a command message came from a TE to the network; the message reference <mr> is returned to the TE.

4.9 Pdu Mode

4.9.1 Read Message with Status

Command:  +CMGL=[<stat>]
Response of TA:
+CMGL:<index><stat>,<length><CR><LF><pdu><CR><LF>...
Action: Reads message with status <stat> and sends them to the TE.

4.9.2 Read Message

Command:  +CMGR=<index>
Response of TA:  +CMGR: <stat>,<length><CR><LF><pdu>
+CMS ERROR: <err>
Action: Reads message in location <index> from preferred message storage and sends it to the TE.

4.9.3 Send Message

Command:  +CMGS=<length><CR>PDU is given<ctrl-Z/ESC>
Response of TA:  if sending succesfull
+CMGS:<mr>
else
+CMS ERROR: <err>

Action: Sends message from the TE to the network.

4.9.4 Write Message

Command: +CMGW=<length>[,<stat>]<CR>PDU is given<ctrl-Z/ESC>
Response of TA: +CMGW: <index>
+CMS ERROR: <err>

Action: Stores message from TE in preferred memory and returns the location <index> to TE.

4.9.5 Send Command Message

Command: +CMGC=<length><CR>PDU is given<ctrl-Z/ESC>
Response of TA: if successful
+CMGC: <mr>
else
+CMS ERROR: <err>

Action: Sends a command message came from the TE to the network and returns the message reference <mr> to the TE.
SECTION 5

IMPLEMENTATION OF REMOTE CONTROL

ATI is the entity that provides service for the execution of the AT and AT+ commands which are generated by a RC application. ATI opens and closes the communication path in lower layers and request AT protocol establishment and release, on the serial line when alerted. The functionality of ATI is to collect the command data from a byte stream, and support a single or multiple commands in a command line. ATI will perform all command parameter checking, perform error handling and inform the Remote Control Application accordingly. When a valid command is received, it will be interpreted, processed and successful completion will be reported to the RC application accordingly. ATI will support all application for Remote Control. For these purposes, a couple of procedures, are provided and can be grouped into the following classes:

ATI related procedures:
- Receiving of command data
- Command line editing
- Parsing, syntax and parameter check comedos
- Processing of commands
- Sending the appropriate response for the command
- Repeat last command
- Handle 10 bit character formats

All AT commands would be supported.

ATI is divided three process. They are: ati lower layer, parser and executer. ATI Lower Layer handles the interface between terminal and MS. ATI lower layer is a serial interface module in ATI. It receives and sends byte from SCI (serial communication interface). Process PARSER handles the parsing of a whole command line received from PARSER. It's main functionality's are: matching the AT(+) command(s), their desired execution modes, their parameters and then requesting the corresponding actions from the EXECUTER. PARSER is the initial state after power-on. After start and initialization of PARSER, PARSER initialize own parameters and send initialization message to EXECUTER process. Then PARSER goes to COMMAND_STATE. In COMMAND_STATE PARSER is not dealing currently with a command line. It indicates that the PARSER is ready to accept a command line from PARSER for parsing.

Process EXECUTER has two state: IDLE and CNF_WAIT_STATE. When the initialize message is received from the PARSER, Executer becomes IDLE. In this state, new command signals may be taken from the parser, also indication messages can come from SL. If the command that will be executed does not need SL interaction, executer state remains IDLE. In otherwise, state goes to CNF_WAIT_STATE. It waits confirmation message from the other process.
5.1 GENERAL REQUIREMENTS

5.1.1 Hard Reset - Power off/on

On power on the MS's remote functionality will be initialised.

5.1.2 Soft Reset - Battery charger / AC adaptor attached and power off/on

The MS can be powered on and off while either the battery is being charged or the AC adaptor is still attached. In both cases no loss of power will take place and hence the MS's volatile memory will not be initialised. Thus when the MS is powered on in these conditions care must be taken to ensure volatile memory is correctly initialised and accessed without causing unnecessary runtime problems. To perform soft reset, ATI will perform a soft reset initialising its RC related settings and parameters.

5.1.3 Startup sequence

Currently the startup sequence of the MS is based around around the validation of the SIM and initialisation of the MS's non-volatile memory, which once performed, acts as a trigger point to request the MS's currently dormant software entities to become active. These triggers are issued in a (sort of) predetermined order which reflects a desired, reasoned sequence of events following power up of the MS.

The SMS submenu cannot be accessed until the main-menu has been presented and MS-User has traversed the MMI and chosen the corresponding submenu option. Thus it is assured that the software entity MMI_SMSPP is already in an active rather than a dormant state when the MS-User request service.

Following power on, the Remote SMS functionality is associated with startup sequence of the lower layers. The remote-User is able to configure the remote interface and perform any non SMS service related features of the remote SMS functionality. Any remote AT+ commands which request access to the MS's SMS service will only be any of its exported services. Until this point, any such request will be rejected and error handling performed accordingly, namely the MS's SMS service is currently not available.

Thus functionality is stand alone and any requests for SMS service will be blocked until such time the MS's SMS service becomes available.

n order for ATI to become operational after power on three steps are needed. The first step is to start ATI so it can make a soft reset as in previous section. The second is the establishment of the communication protocol on serial communication module. The third is the establishment of serial line protocol on ATI. After these three steps are realised ATI is ready to receive commands, process them and send their responses.

5.1.4 Detection of Remote Cable Connected to MS

No notification of the cable's status shall be provided to the MS-User. Assuming prior configuration of SMS messages to be re-routed and/or message indications given, namely via the AT+ command +CNMI, the cable's status is used to
determine whether or not received SMS MT messages are stored on the SIM or routed/indicated to the Remote PC.

The remote cable can be detected at any time by the remote-User. When this occurs, either rerouting or message indication configurations may or may not be configured. If no such configurations are present, the detachment of the cable is no significance. However, if such configurations are present, the MS must suspend the requested routing/indications and reconfigure the MS to store all messages on the SIM.

The software entity, MN-SMS, will get information from the lower layer software entity, DEVCO concerning the cable's detached status. This will cause the MN-SMS layer to reference the default section of the COMA table which indicate that all SMS messages are to be stored on the SIM.

The remote cable can be attached at any time by the remote-User. When this occurs, re-routing of messages and indications may or may not be suspended. If no suspension is active, the attachment of the cable has no significance. However, if re-routing has been suspended, cancel the suspension by recognition the MS to stop storing all messages on the SIM and resume the requested routing/indications.

The software entity, MN-SMS, get information from the lower layer software entity, DEVCO, concerning the cable's attached status. This will cause the MN-SMS layer to reference the configured section of the COMA table which will indicate that either all SMS messages are to be stored on the SIM, namely its default setting, or the desired routing/indications as configured via the AT+ command +CNMI.

5.1.5 Session control of MS's SMS service

It is stated that the MS's SMS service is single-user, meaning that the local MMI and remote SMS must access the service independently and mutually exclusively.

5.1.6 SMS session establishment and termination

All of the exported SMS services will be made available once a session has been established by a client. The MS's SMS service will only allow a session to be established when all of its exported services can be supported, namely when the library is in an active and operational state. An established session can only be terminated on request by the client.

The means to request session establishment and termination will be appropriately implemented. It is not foreseen that any necessary housekeeping operations will need to be performed by either the client or the exported service(s) once a successfully session has been terminated.

The local MMI will request an SMS session when the MS the MS-User enters the SMS sub-menu by whatever means. Once the session request has been granted, the local MMI will request release of the session when, for whatever reason, the MMI return either to the main menu or the main menu or the Idle screen. The local MMI's SMS session will be for as long as the MS-User remains within the SMS submenu.

The remote SMS will request an SMS session whenever an ATI command is entered requiring the appropriate exported SMS service. When the SMS service has
been performed, the remote SMS session will be terminated. The remote's use of the SMS service will be fleeting and for only as long as is necessary.

5.1.7 SMS session time-out mechanism

Because the SMS session is single-user orientated it is important that a session must be released once it has been established by either the local MMI or remote SMS. If not, the SMS service remains blocked for the other user.

The local MMI employs a handset timeout which causes the MMI screen to return to idle, this will insure that the local MMI always releases the SMS session. Therefore, there are no foreseen scenarios in which the Local MMI could not release the SMS session.

The remote SMS will guarantee to perform an orderly release of the session after the last requested service has been performed. Thus the remote SMS will always request session termination. Because of this it is deemed not necessary for the remote to employ a timeout mechanism to prevent extended session access.

5.1.8 The SMS session rejection

It is possible for the MS-User to be using the SMS submenu when a session request is made by the remote. In this case, the local MMI has control of the SMS session and the remote's request will be rejected. Note, If the remote-User enters a text message for transmission and the associated ATI command is rejected because the Local MMI has control of the SMS session, the A\ command, namely Repeat Last Command, allows the remote-User to be able to repeat the command without having to re-enter the text.

Once an SMS session has been established all the exported services will be operational. However, the MS's SMS service will only allow a session to be established when it is an active, operational state. This state will only be achieved when certain startup criteria are met. Part of this criteria is that a validated SIM has been inserted and authenticated, namely its correct password has been entered. Thus if the remote SMS attempts session access before a SIM has been inserted, validated or authenticated the request will be rejected.

It is necessary that access to services like sending an SMS message needs to be controlled via a session, however, it is also assumed that not all SMS exported services need this degree of control. The fact is that certain SMS exported services need access control rather than controlled access to the whole MS's SMS service. For this reason, necessary and appropriate SMS exported services will be made available without the need for a session to be first established.

There is commonality between the requirements of the Local MMI and the remote SMS with regards to the handling and processing of SMS MO and MT messages, whether it be assembling a PDU or simple parameter checking and message validity. It is deemed necessary that this commonality shall be exploited thereby removing any duplication of effort. The MS's SMS exported services are the means by which this commonly can be exploited. Local MMI provides the MS-User with a simple option list of control values to choose from. Not all possible values are available nor are all SMS control parameters presented. The remote SMS allows the MS-User to directly configure several parameters without restriction, using the associated AT+ command.
5.1.9 Settings Maintenance – Remote SMS control parameters & Profile 0

The remote SMS will maintain in the MS’s volatile memory variables which relate to the following SMS control parameters, each of which can be configured by the remote-User using the appropriate AT+ commands

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCA</td>
<td>Service Control Address</td>
</tr>
<tr>
<td>TOSCA</td>
<td>Type Of Service Centre Address</td>
</tr>
<tr>
<td>F0</td>
<td>First Octet of any SMS Protocol Data Unit</td>
</tr>
<tr>
<td>VP</td>
<td>Validity Period</td>
</tr>
<tr>
<td>PID</td>
<td>Protocol Identifier</td>
</tr>
<tr>
<td>DCS</td>
<td>Data Coding Scheme</td>
</tr>
</tbody>
</table>

The remote SMS will only support one profile, namely profile 0. The AT+ command, +CSAS, instructs the MS to save in the MS’s non volatile memory the current values of the remote SMS’s control parameters, which are held in volatile memory. The SMS control parameters stored as Profile 0 will be totally independent of the settings which are accessible via the SMS submenu and are stored in the SIM and/or the MS’s non-volatile memory.

Note, the AT+ command, +CREAS, causes the MS to copy the stored values of Profile 0, held in the MS’s non-volatile memory, to the Remote’s volatile SMS control parameters.

On power-up or a change of SIM, the remote SMS will assign to each of the above SMS control parameters the appropriately defined GSM default values, namely

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>17 integer, i.e. SMS-SUBMIT</td>
</tr>
<tr>
<td>VP</td>
<td>167 integer, i.e. 0..255</td>
</tr>
<tr>
<td>PID</td>
<td>0</td>
</tr>
<tr>
<td>DCS</td>
<td>0</td>
</tr>
<tr>
<td>CSA</td>
<td>nothing, i.e. '***'</td>
</tr>
<tr>
<td>TOSCA</td>
<td>129, i.e. no leading + character</td>
</tr>
</tbody>
</table>

5.1.10 Routing Control

The Remote SMS has the capability with the AT+ command, +CNMI, to instruct the MS to either not store certain classes of SMS MT messages on the SIM and instead route them through for transfer over the external serial line, or store certain classes of SMS MT messages on the SIM but provide notification as and when such messages arrive. The notification includes and “index” parameter which can be referenced in other AT+ commands to refer specifically to the stored message on the SIM. The characteristics and properties of the required routing and indication behaviour are described in the following sections.

The AT+ command, +CNML, instructs the MS to re-route either SMS-DELIVER (class 0,1 and 3 or class 3 only) or STATUS-REPORT PDU’s. The command also instructs the MS to provide notification of stored delivery of all classes of SMS-DELIVER PDU’s. These are the basic requirements.

The MS’s SMS service will export an appropriate SMS service which will configuring the COMA table to the correct settings as instructed in the AT+ command, +CNMI.
Once the requested routing has been configured, the delivery of SMS-DELIVER and STATUS REPORTS must be processed accordingly. Messages which are configured to be routed through the MS must not be stored but passed onto the appropriate software entities. Such messages should be routed to the remote SMS not to the Local MMI, regardless of any session control.

An Indication should contain the correct “index” or page number in order that the message can be uniquely referenced by the remote-User in other AT+ related commands. Again such Indications should be passed onto the remote SMS not the Local MMI, regardless of any session control.

The delivery of SMS MT messages is unpredictable and totally asynchronous to the controlled access of the MS’s SMS service as made available to both the Local MMI and remote SMS. Therefore, routing of messages and indications will be performed regardless of who is currently engaged in a session with the MS’s SMS service. Similarly, if the remote SMS is currently engaged in a session the routing of messages and indications will still take place, regardless if a requested service has still not yet completed.

It is assumed at this stage in the development process, that no flow control is required between the MS’s SMS service and the remote SMS with regards to the asynchronous delivery of either re-routed SMS messages or message indications. The stated requirement is to minimize any possible loss of such message indications, by whatever means.

The following is a list of all the SMS related indications, namely text messages, icons and ringer management.

A suitable text message informs the MS-User of the following conditions
- The SIM is now full
- The SIM has overflowed
- A message or messages have been delivered

The SMS related icons are as follows,
- Static Envelope Icon – At least one message is stored on the SIM
- Flashing Envelope Icon – At least one message remains unread

The MS’s Ringer is used as follows
- It is rung whenever an SMS message is delivered and attempted storage on the SIM, including voice mail

The above are still relevant when the MS supports remote SMS access. There will be interworking between such routed messages/indications and the remote SMS. The generation of such indications occurs as and when the associated events actually happen in real-time rather than at some later predefined state or condition. In summary, the indications are generated independently of whom is currently engaged in a session with the MS’s SMS service.

5.1.11 AT1 Command Management and Responses

AT1 will get the command data from a byte stream. The command line is extracted from the byte stream by searching for the command line termination
character. The command line termination character can be changed by setting the S3 parameter in the V.25ter specification. ATI will also handle the response formatting and command line editing characters when interfacing with the TE. These characters are also affected by the settings of the S4 and S5 parameters in the V.25ter specification respectively.

A special command to be taken care of is the "A/" (Repeat Previous Command line), which causes the ATI to re-execute the previous command line. The whole command line must be buffered without a change, in case the next command may be the "A/". For the rest of the commands, the command line must be started with the Command Line Prefix "AT" and it may consist of a single command or multiple commands and also some white space used to increase readability. If the command line consists of multiple commands, the basic commands which are specified in the V.25ter may come one after each other without having a special delimiter between them. Each extended command must be prefixed by a "+" and must be delimited by a ";" from the other commands as specified in GSM 07.07.

The command line must be cleared from the white space characters and searched for a matching command from left to right. When a matching command is found, the desired action mode for the command must be identified and parameters for the command, if any, must be extracted. The parameters of a command are expected to be separated by commas, and the ignored parameters of a command are assigned to their current values. If the desired action for the command requires parameters, but no parameters are given, the command should be executed with current parameters if possible.

In case of an error which may be caused by:

- The command string can not be matched,
- The action for the command is not supported,
- Mismatch in number of parameters,
- Mismatch in type of parameters,

The error causing command and the rest of the command line must be ignored and the corresponding error message must be returned to the TE. Some of the commands support both Text and PDU modes and the expected number and type of the parameters of a command may change based on the current mode. This must be taken care of while checking for the number and type of parameters of a command which support both Text and PDU modes. For the commands, "+CMGS", "+CMGW" and "+CMGC", the command is ended with a <CR> character, followed by the data rate related to the command.

In these cases, after receiving the <CR> character, ATI must enter a state to accept the command data, which can be either text or pdu string. Upon receipt of the <CR> character, ATI must enter a state to accept the command data and while receiving the command data, ATI will send a four character sequence CR><LF><greater_than><space> after a command line is terminated with <CR>, as defined in sections 3.5.1 and 4.3 of the GSM 07.05 specification. The command data is ended with a <Ctrl-Z> character, or the whole command can be cancelled by giving the <ESC> character.
Result codes for the commands, may be suppressed or enabled by setting the value for the Q command and the response format can be changed by setting the value for the V command.

5.1.12 F0 Handling

If the RC application is in the PDU mode it is assumed that the PDU's which are being sent to ATI are complete and correct so no controls will be done in ATI. If the applications in text mode then it is ATI's job to determine the F0 octet by comparing the F0 value in the volatile profile 0 and the possible values of F0 that can be according to the command. Possible commands and their interactions with F0 in text mode are given below.

- When writing an SMS message to the memory only SMS SUBMIT or SMS DELIVER types are supported. So the message identifier field in F0 which is in volatile profile 0 is checked. If it is not SMS-SUBMIT or SMS-DELIVER then the message type will be considered as SMS-SUBMIT.

- When sending a message to network, message identifier field in F0 will be set to SMS-SUBMIT regardless of the value in F0 which is in volatile profile 0.

- When sending and SMS-Command Message to network message identifier field in F0 will be set to SMS-COMMAND regardless of the value in F0 which is in remote SMS settings in volatile memory.

5.1.13 Unsolicited SMS Messages and Indications

ATI will not support any buffering mechanisms for keeping unsolicited messages and indications. When an unsolicited SMS message or an indication is in ATI's message queue ATI will behave according to its existing state. If it is currently processing a command received from the RC application first it will finish processing this command after that it will route the unsolicited SMS message or indication to the RC application. If nothing is being processed in ATI then ATI will directly route the unsolicited SMS message or indication to the RC application

5.1.14 Error Mode

There are three different error types. The error types are given below:

i) +CMS ERROR: <err> indicates an error related to ME or network, for the commands that are defined of GSM 07.05.

ii) +CME ERROR: <err> indicates an error related to ME for the commands that are defined of GSM 07.07. 
<err> values are defined in section 8.2.2 of GSM 07.07.

iii) ERROR is returned normally when error is related to syntax for the commands that are defined in V.25ter.
Error message format is affected by using V (DCE Response Format) and +CMEE commands)Report Mobile Equipment Error).

V command controls the basic commands, where +CMEE controls the extended commands that are defined of GSM 07.07.

+CMEE command disables or enables the use of result code +CME ERROR: <err> as an indication of an error relating to the functionality of the ME.

+CMEE command is defined in section 8.1 of GSM 07.07.

V command determines whether result codes are transmitted in numeric format or alphabetic (verbose) format. The text portion of information responses is not affected by this setting.

5.1.15 Remote Settings Maintenance

Profile structure will be kept in RAM. Only profile 0 is supported. The profile structure includes the following the parameters:

Service centre address <sca>
Type of Service centre address <tosca>
First octet <fo>
Validity period <vp>
Protocol identifier <pid>
Data coding scheme <dcs>

Profile structure is saved to non-volatile memory by using the command +CSAS and is restored to activate memory (RAM) from non-volatile memory by using the command +CRES.

5.1.16 Assembling PDU

The SMS messages which are being exchanged between ATI and SMS Service Library are in PDU format. When the RC application is in text mode, commands and responses which contain SMS messages must be assembled or disassembled. While receiving a command from the RC Application including an SMS message ATI will assemble the PDU by converting the SMS control parameters (refer GSM 07.05 3.1. Message Data Parameters) and message body into PDU format. If some of the optional parameters are missing in the command line ATI will fill these parts accordingly. While sending a response to the RC application including an SMS message first ATI will disassemble the PDU into its constituent parts, which are text body and SMS control parameters.
5.2 ARCHITECTURE

The purpose of the CAF architecture is to provide a complete set of mechanisms that ease the development of application software for cellular terminals. The shift from the monolithic MMI software as used by the HTxx11 series of products to a framework-oriented approach is significant and driven by both restrictions implied by monolithic designs, and new software requirements, such as remote controllable features.

The Common Application Framework can be split into two basic areas of functionality that are completely independent of each other: A service interface providing access to several facilities needed by applications (such as phonebook data or protocol stack resources), and a User Interface Management System (UIMS) that controls local application software. The rest of this overview introduces the application model used throughout CAF and summarizes the features of the framework's two major components.

5.2.1 Application Model

The CAF architecture is designed to support any kind of applications. Thus, both local MMI software as well as remote application components are served. Local applications can be divided into two separate areas of functionality each of which has well-defined responsibilities and uses a specific component of the CAF: The presentation part of the application is used for presenting data to the user and manages keyboard input. It is coded using so-called “characteristics lists” which are interpreted by the UIMS at run-time, and callback routines triggered by the UIMS while interpreting such characteristics lists. The logic part implements application functionality that has no user interface aspects controls the flow of execution, and communicates with other DTe1 components.

As the above figure suggests, presentation and logic part of an application are not completely decoupled: The callback routines implemented by the application contain both sections dedicated to the presentation part, and sections dedicated to the logic.

![Diagram](image)

Figure 5.1 Local Application Software Accesses CAF
part. DTel components serving remote applications (the AT command interpreter is an example of such a component) do not implement a local presentation part and therefore use features of the service interface only.

5.2.2 Service Interface

The service interface allows application software to access functionality of the DTel software conveniently. It provides a set of libraries dedicated to various aspects of the protocol stack or other parts of the system, an access management that allows collision-free usage of these facilities by concurrent applications, and a variety of utility mechanisms that simplify the task of enhancing existing service libraries or adding new ones.

The philosophy of a service library is to offer a set of atomic functions that allow applications to make use of certain DTel features. The functionality of other interfaces — e.g., those offered by the protocol stack are nevertheless not mirrored, instead, a service library encapsulates such interfaces and acts as a mediator between an application and one or more DTel components. Together with a well-defined access management, this approach allows applications to use DTel features conveniently and consistent, without being aware of the underlaying mechanisms.

5.2.3 User Interface Management System

The User Interface Management System — also called MMI Kernel — allows to build up applications without the need to care about details like input event management, display handling, or application interaction. The UIMS provides a so-called characteristics lists interpreter which parses application-defined data objects specifying the behaviour of the presentation part of an application. In addition, the UIMS implements a set of utility functions that assist in programming such presentation parts.

In order to further simplify the task of creating application software, the Local Application Development Toolkit (LADT) offers a set of tools that allow the automated generation of application skeletons.

5.2.4 Service Interface Architecture

The service interface consists of a collection of components that provide easy-to-use access to DTel functionality. These components are:

- The Service Library Access Manager (SLAM).

A set of service libraries dedicated to the following areas of functionality:

- PLMN management
- Speech, data and fax call management, call-related supplementary services
- Handling of call-independent supplementary service operations
- Short message service (point-to-point) access
- Cell broadcast message handling
- Phonebook resource access
- Security services (e.g. PIN handling)
Voice recognition facility access
Audio resource access
Mobile status data access

A Queue Manager (QMAN) that is used by the service libraries in order to queue application requests for later processing.

5.2.5 Mechanisms

While most of the DTel components communicate with each other by means of a mailbox system (that is, they exchange messages), the service interface uses a functional approach. All services are implemented as regular C-functions; asynchronous events are reported by using callback routines supplied by an application. Basically, two types of asynchronous events can be distinguished:

Services that complete asynchronously do not require any kind of session management or service subscription. Each library takes care about handling multiple requests properly and informs the requesting application about the completion. Notification about unsolicited events (such as the arrival of a short message) or events that are of interest to more than just the application which originally has caused them (e.g. the modification of shared data, such as a phonebook entry) require subscription. The application uses the access management functionality — see next section — to subscribe to the notification mechanism, specifying which event category it is interested in and providing the address of a callback routine which will be called when such an event is encountered.

Both type of asynchronous events cause callbacks to be triggered that use two formal parameters: A 32-bit code specifying the reason for the callback routine activation (such as "Asynchronous operation completed successfully"), and an (optional) pointer to a data structure providing more information about the event. The structure’s internal layout can vary and depends on the service function.

Because callback functions execute in the context of the service interface, it is important to consider some implementation restrictions when coding such routines. These restrictions are:

- Data passed to a callback routine is volatile. As soon as the callback relinquishes control, the service library is free to re-use or deallocate memory used for passing the data. Thus, callback routines should always copy the received information to private memory.

- Callback routines are not allowed to run for a longer period of time. If the application has to perform lengthy actions as a result of the activation of a callback routine, it has to take the appropriate actions to save the data passed and to re-invoke itself upon the next scheduled entry of the associated task.

- Callback routines must not call service library functions.
5.2.6 Service Library Access Management

The Service Library Access Manager (SLAM) is a special component that provides an interface to the CAF service libraries. The first base level of the SLAM will support access to the event indication mechanism of the service libraries only, future base levels may add further functionality, such as providing information about service library status and capabilities or an authentication facility.

The event indication mechanism ensures that applications will be notified when some unsolicited event happens. It also ensures that events being of interest to more than one component (so-called “public” events, e.g. the modification of a data item shared by several applications) are distributed accordingly. As a result, the SLAM approach also allows an application to rely on “echos” of certain operations. For example, if an application has subscribed to the event category “phonebook record has changed”, and then modifies a phonebook entry, it will receive a dedicated indication informing it about the change caused by itself. This mechanism can also be used to simplify the design of applications.

Each application that needs notification specifies information about the category of event it is interested in (examples for categories are “Short message delivery”, or “Mobile terminated speech call arrived”) and registers with the SLAM. A service library having processed an unsolicited or “public” event will pass dedicated event information to the SLAM, which then dispatches the information to all subscribers of the respective event category.

![Figure 5.2 Distribution of Events Using the SLAM](image)

The approach described above ensures that only the SLAM stores information about applications having subscribed event notifications. It also means that notifications about unsolicited or “public” events are distributed to all applications being on the distribution list.

Under certain circumstances, an application may be notified about events it is not interested in. For example, an event “new cell broadcast message received” is not of interest to an application in case it hasn’t subscribed the message identification
number of the message received. Note that it's always the task of the application to filter the incoming events and to process only those indications that really matter.

5.2.7 Service Libraries

Each service library interface consists of a set of functions that can be called by application software. Basically, there are no access restrictions although service library functions that initiate operations may succeed or fail due to resource problems. Thus, service functions return information about the requested operation's status. Supported values are:

- "OK" — The operation has successfully completed (synchronous services only).
- "QUEUED" — The operation could not be initiated because the service library's resources are temporarily exhausted. (E.g., the library is currently serving some other application.) However, the request has been queued (using the CAF's queue manager) for later processing.
- "RUNNING" — The operation has been initiated but the result is not yet known because of the operation's asynchronous nature. The library will call an application-defined callback routine when the operation has completed.
- "FAILED" — The operation could not be initiated/completed for some reason.

Service functions that are called by an application may complete asynchronously; the library will execute a callback routine specified by the application as soon as the operation has completed.

Figure 5.3 Handling of Requests that Complete Asynchronously

The mechanism described above is only used for handling requests that complete asynchronously and do not cause results that may also be of interest to other applications. For example, the transmit status of a short message would be indicated to the requesting application in the way described above. However, the modification of a phonebook record would be indicated using the SLAM, because such a request has presumably to be handled by other applications also. Service libraries can call routines of other service libraries and are also allowed to subscribe to event indications using the SLAM.
5.2.8 Utility Functionality

The service interface section of the CAF also implements a set of utility routines that are shared among the service libraries. Although the utility functionality is basically intended to assist in the construction of service libraries, applications are also allowed to call the respective functions.

5.2.9 UIMS Software

The User Interface Management System (most often called “the MMI Kernel”) provides the basic infrastructure for MMI applications. The following list outlines this system:

- All applications use one or more so-called “contexts”. A context is a run-time object that defines particular aspects of the behaviour of an application. Contexts have priorities, which allows one context to take precedence over another context. The context owning the highest priority is called the foreground context; the application using the foreground context controls the display. (As soon as a foreground context gets deleted, the context with the next lower priority becomes the new foreground context.)

- When using the UIMS, applications are just collections of callback routines and characteristics lists (see next topic). Callback routines are triggered by the UIMS as the result of a so-called event (Examples for events are the arrival of a primitive, a key press, etc.), or upon creation and/or deletion of a context. Events are always dispatched to an existing context (but not necessarily to the foreground context). The context receiving an event is called the active context.

- Each context owns a characteristics list (also often referred to as scenario.) The characteristics list describes how a context should react upon delivery of a particular event (e.g. what callback routine should be called) and how the display should be organized. In addition, a characteristics list defines its successors, that is, it determines other characteristics lists that replace itself when some event arrives. Characteristics lists are written in a simple macro language and are transformed into C-code at build time, using a so-called Scenario Compiler.

- The UIMS software allows applications to display text by using so-called labels instead of actual strings. Labels are numeric figures being independent from a particular language. This means that applications can display text without taking care about the language to be used. Turning labels into actual text (using the
language selected by the user) as well as decompressing and formatting the text is the task of the Kernel. It also offers access to the pictogram line of the display and different text formatting options (e.g. centering text on a display line, display blinking text, or display text as a so-called “ticker tape”) as well as text browsing facilities. Also, all text input is managed by the UIMS: Any application that requires user input uses the editors provided by the Kernel software. There are a number of editors dedicated to different tasks, e.g. editing a Short Message, entering a phone number, or adding a phonebook entry.

The following additional functionality can be accessed by utilizing a special convenience routine interface of the UIMS software:

- **Keyboard Handling** — This part of the software is able to translate key codes (transmitted by the DTel KBI) into *virtual key events*, like “clear key pressed for a longer period of time”. MMI applications rely on virtual events and never access KBI key codes directly.
- **Sound Management** — The interface offers routines being able to generate key beeps and ringing melodies.
- **Timer Handling** — The Kernel provides access to the AEROS timer resource. Timer expiration is signalled as an event delivered to a context.
- **The functionality described above is dedicated to applications only, that is, service libraries are not affected.**
- **The UIMS software has been designed to suite the user interface needs of the HT1xx20 series of products. Future requirements may require the introduction of new concepts and strategies. (An example would be the introduction of a data pipe that allows transparent passing of information between applications without the need of explicitly coupling them.)**
RESULTS AND RECOMMENDATIONS

The goal of the project Remote Control is to provide a service interface to a TE enabling the TE to use services of the MS. For offering a generic interface the project will use the means standardized in GSM 07.07 and GSM 07.05 for implementing this interface. By using a standardised mechanism the interface is open to any application which are able to use this mechanism. AT command set is used by supports MS functionality challenging.

In the approach taken in this document is depicted. First, the services which must be offered by a MS to remote applications are analysed. A remote application is an application using the services offered by MS. The application may e.g. run on a PC connected to the MS by a communication interface. Primarily it states the services offered by GSM 07.05 and GSM 07.07 and then explains the extended AT-Command set describing each command in detail and grouping them according to their functionality's. Second, as a result of this analysis an ideal software architecture has been stated and compared with the existing one in order to point out the changes which have to be implemented.

Between a terminal and a GSM phone an RS-232 connection is established via RS cable. With any terminal program (HyperTerminal, VT220 emulation program), it can be reached to a GSM phone and all the requirements made by GSM can be handled. The commands that handles GSM applications can be found in GSM 07.05 and GSM 07.07 documents in detail.

AT-commands explained in standards provide sending and receiving SMS message, reading and writing of phonebook, listing and changing network, changing PIN-PUK number of GSM phone, calling setup and other call features. To provide these, the related software is prepared for AEG-NETAS GSM phone. This software consists of three parts:

♦ Serial Communication Protocol (ATILLA)
♦ The parsing of commands (PARSER)
♦ The functions that handles commands (EXECUTER)

ATI is the entity that provides service for the execution of the AT and AT+ commands which are generated by a RC application. ATI opens and closes the communication path in lower layers and request AT protocol establishment and release, on the serial line when alerted. The functionality of ATI is to collect the command data from a byte stream, and support a single or multiple commands in a command line. ATI will perform all command parameter checking, perform error handling and inform the Remote Control Application accordingly. When a valid command is received, it will be interpreted, processed and successful completion will be reported to the RC application accordingly. ATI will support all aplication for Remote Control.
This software (ATI-MODULE) is designed with the integration of other modules in AEG-NETAS hand-phone. All software is run by loading the software to flash memory in AEG-NETAS GSM. In the progressing phase of project DATA-FAX property will be added.

The aim of this project is to provide GSM uses a way of sending / receiving messages and data via wireless systems. By this way, the GSM is connected directly computer and a variety of data applications such as internet, fax, making call, sending and receiving messages can be performed. This application is done by embedding the software interface in the created module ATI and therefore prevents the user from wasting lots of many in PCMCIA card (this interface card gives the same service with the interface software).

Although the ATI module is designed for implementing the remote control system in AEG handy phones, it can be adapted to other systems’ products by updating the interface part of the software between the lower layers. This update could easily be done for the reason that this software has already be implemented according to ETSI standards.

Finally the software developed in this project provides the advantage of using the data applications independent of place. This product, which can also be applied in cars by adapting it to the on-board computer system, will be very popular in the future.
REFERENCES

[1] ETSI  AT command Set for GSM Mobile Equipment (ME), (GSM 07.07)


[6] ETSI  Radio Link Protocol (RLP) for data and telematic services on the Mobile Station - Base Station System (MS - BSS) interface and the Base Station - Mobile Service Switching Centre (BSS - MSC) interface (GSM 04.22; Phase 2).

[7] ETSI  European digital cellular telecommunications system (Phase 2); Alphabets and language-specific information (GSM 03.38).


APPENDIX A

SDL DIAGRAMS

ATI SDL diagram is prepared by SDT tool. All diagrams are listed the following pages.
edure reset_parser

sets lexical analyzer state to initial state, and currPtr to the beginning of the cmdLineBuff.

Reset lexical analyzer states.

\[
\text{ati}\_\text{parserData} = \text{ati}\_\text{currPtr} := \text{ati}\_\text{data} = \text{ati}\_\text{cmdLineBuff}
\]

\[
\text{ati}\_\text{parserData} = \text{ati}\_\text{ParseLexicalState} := \text{START\_STATE}
\]
match_command_parameters

1. i = 0
2. If i < catl_kernelData.numOfParams
   - If (ati_parserData.cmdIndex == CSMP_CMD_BASIC_CMD_TABLE_SIZE) & (i == 1)
     - ReturnValue := MATCH_CMD_PARAMS_OK
   - Else
     - If i < 1, Go to 3
     - Else, Go to 4
3. i = i + 1
4. If i < strlen(ati_kernelData[paramBuff(i)])
   - paramSize := return
     - If paramSize = 0
       - ReturnValue := MATCH_CMD_PARAMS_ERR
     - Else
       - If (paramTypes[i] & CHECK_MASK) = CHECK_MASK
         - Go to 5
       - Else
         - Go to 6
5. Else
   - If (ati_kernelData[paramBuff(i,0)] = STRING_DELIMITER) & (ati_kernelData[paramBuff(i,paramSize-1)] = STRING_DELIMITER) & (paramSize = 1)
     - ReturnValue := MATCH_CMD_PARAMS_ERR
   - Else
     - ReturnValue := MATCH_CMD_PARAMS_ERR
6. Go to 5


```c

dure match_command_parameters

the individual parameters

= parameter string.

's CmdParamBuff' array and

'CmdParams'.

```
The individual parameters are parameter string: "CmdParamBuff" array and 'CmdParams'.

```
true

2

false

temp != CMD_DELIMITER

ati_kernalData withSemiColon := TRUE

ati_kernalData withSemiColon := FALSE

i := 0
j := 0

mem_alloc [paramSize+1]

result := ati_kernalData[paramBuff0]

false

< cntindex

true

false

ati_kernalData[numOfParams := 1
ati_parserData currPtr := ati_parserData currPtr + cntindex
ati_kernalData[paramBuff0, paramSize] := '0'

ati_kernalData[paramBuff0, j] := temp
j := i + 1

l := i + 1

((temp > 47) && (temp < 58)) || (temp = 35) || (temp = 42) || (temp = 43)

returnValue := MATCH_CMD_PARAMS_ERR
```
null match_command_id

if (loopCount != 0) {
    ++loopCount;
    ... conditions...
}

int startFlag, endFlag; curIndex = startIndex;

while (true) {
    if (matchCmdId == MATCH_CMD_ID_OK) {
        return MATCH_CMD_ID_OK;
    }
    else if (matchCmdId == MATCH_CMD_ID_ERR) {
        return MATCH_CMD_ID_ERR;
    }
    ... conditions...
}

if (foundFlag == TRUE) {
    ati_parserDataCmdIndex = curIndex;
    ti_parserDataCmdPtr = ((currCmdTable[curIndex].execMode) >> 4) & 0x07;
}

selectedTable = EXT_CMD_TABLE;

parserData = DT_EXT_CMD_STATE;

if (ati_parserDataLexicalState == GET_BASIC_CMD_STATE) {
    ... conditions...
}

return MATCH_CMD_ID_OK;
procedure match_command_id

maxLoopCount, loopCount int;
tartIndex, endIndex, currIndex int;
foundFlag Boolean;
int;

if the given search table, it tries to match an AT command text
in cmd line buffer, searching from left to right,
ing the Cmdid of the longest match or
r, if no command could be matched.
dates the 'Cmdid' variable and
ates the 'Lexical State' as one of the
ng according to the type of the
matched:
SOT_BASIC_CMD,
SOT_EXT_CMD,

selectedTable = EXT_CMD_TABLE
True

(ati_parserDatacurrPtr)[0] = '1'
True

return(MATCH_CMD_ID_ERR)
ati_parserDatacurrPtr++
endIndex = BASIC_CMD_TABLE_SIZE - 1

endIndex = EXT_CMD_TABLE_SIZE - 1

"Restrict the search length
to the minimum one of the
length of the command line
buffer or maximum command length"

strlen(char *)ati_
parserDatacurrPtr > MAX_CMD_LEN
True

maxMatchLen = strlen(char *)ati_
parserDatacurrPtr)

maxMatchLen = MAX_CMD_LEN

1


Given a command table, this procedure checks the execution mode of the matched command.

Procedure: `match_command_exec_mode`

1. Initialize `currCmdTable` to `ati_extCmdTable` if `ati_parserDataExtCmdState` equals `GOT_EXT_CMD_STATE`.
2. Otherwise, set `currCmdTable` to `ati_basicCmdTable`.
3. Check if `currCmdTable` is `null`.
   - If `true`, set `ati_parserDatacmdIndex` to `null`.
   - If `false`, check if `ati_parserDatacurrPtr` equals `null`.
     - If `true`, proceed to the next step.
     - If `false`, set `ati_parserDatacmdExeMode` to `QUERY_SUPPORTED`.
6. Increment `ati_parserDatacurrPtr` by 2.
7. Check if `ati_parserDatacurrPtr` equals `null`.
   - If `true`, set `ati_parserDatacmdExeMode` to `QUERY_CURRENT`.
   - If `false`, set `ati_parserDatacurrPtr` to 1.
8. Set `returnValue` to `MATCH_CMD_EM_OK`.

Procedure eliminateSpaces

Eliminates the space characters in the cmdLineBuff, which are not

Discard white spaces
in the command line buffer

"", (i.e. stringMode==FALSE). It updates the cmdLineBuff,

int;  
emptyIndex int;  
StringMode Boolean;

loop

Index := 0

Mode := FALSE

loop

CmdLineBuffLen := emptyIndex

CmdLineBuff[i] = FALSE

CmdLineBuffLen > 32

CmdLineBuff[i] = CmdLineBuff[emptyIndex]

emptyIndex++

stringMode := FALSE

stringMode := TRUE

stringMode := TRUE

loop

loop
```c
const int BASIC_CMD_TABLE_SIZE = 28;
const int EXT_CMD_TABLE_SIZE = 44;
const int MAX_CMD_LEN = 6;
const int NUM_OF_PDU_CMDS = 4;
const int MAX_CMD_LINE_LEN = 80;
const int MAX_PARAM_STRING_LEN = 80;
const int MAX_NEXT_CMD_TEXT_LEN = 331;
const int MAX_NEXT_PDU_MSG_LEN = 353;
const int MAX_NEXT_TEX_MSG_LEN = 161;
const int MAX_NEXT_TEXT_HEX_LEN = 321;
const int MAX_NEXT_TEXT_8BIT_LEN = 281;

const int CHECK_MASK = 0x01 /* check least significant bit */;
const int QC_FLAG = 0x01 /* uses least significant bit */;
const int QS_FLAG = 0x02 /* uses 2nd least significant bit */;
const int ASS_FLAG = 0x04 /* uses 3rd least significant bit */;
const int ASS_DEF_FLAG = 0x08 /* uses 4th least significant bit */;
const int QC_BIT_INDEX = 0;
const int QS_BIT_INDEX = 1;
const int ASS_BIT_INDEX = 2;
const int ASS_DEF_BIT_INDEX = 3;
const int ONE_BYTE_CMD = 0x10;
const int TWO_BYTE_CMD = 0x20;
const int THREE_BYTE_CMD = 0x30;
const int FOUR_BYTE_CMD = 0x40;
const int FIVE_BYTE_CMD = 0x50;
const int SIX_BYTE_CMD = 0x60;
const int CMD_DELIMITER = 0x3B /* "" character used to delimit AT+ commands */;
const int STRING_DELIMITER = 0x22 /* "" character used to delimit string exprns */;
const int PARAM_DELIMITER = 0x2C /* "" character used to delimit parameters of a cmd */;
const int EQUALSIGN = 0x3D;
const int QUESTION_MARK = 0x3F;
const int CMS_CMD = 0x80;
```
NEWTYPE CmdExecModeType
  LITERALS QUERY_CURRENT, QUERY_SUPPORTED, ASSIGNMENT
ENDNEWTYPE CmdExecModeType;

NEWTYPE LexicalStateType
  LITERALS START_STATE, GOT_BASIC_CMD, GOT_EXT_CMD
ENDNEWTYPE LexicalStateType;

DCL CmdLine Charstring;
DCL CmdLineLen CmdLineLenType;
DCL MaxCmdLineLen CmdLineLenType;
DCL CurrPtr Charstring;
DCL CmdId Int;
DCL CmdExecMode CmdExecModeType;

DCL CmdParamBuff Array(MaxNumOfParams, Charstring);
DCL NumOfCmdParams Int;

DCL CmdData Charstring;
DCL CmdDataLen Int;

DCL CmdLineTermChar Character;
DCL Mode ModeType;
DCL LexicalState LexicalStateType;
DCL Result ResultType;
DCL NextMode ByteInterfaceModeType;
DCL ExecFlag Boolean;
DCL ResultText Charstring;
DCL CommandStatus CmdStatusType;
DCL CommandResponse ResponseType;
DCL Dcs Boolean;
DCL ChSet ChSetType;
procedure writeResultCodeSuppMode

{

genericFunction one (&at_kernelData, ResultCodeSuppFlag, ATI_RESET_2, V25TER_ERR)

}
procedure writePhbookEntry

A phonebook entry in location number <index> in the current
phonebook memory storage selected with +CPBS. Entry fields written
phone number <number> and text associated with the number.

3

\[ \text{text} \triangleq (\text{char *} \text{mem_alloc}) \]
\[ \text{ati}_\text{kernelDataLength} \]
\[ \text{ati}_\text{kernelDataNumOfParams} \triangleq 3 \]
\[ \text{lenOfStr} \triangleq \text{strlen(char *) ati}_\text{kernelDataParamBuff[3]} \]
\[ (\text{lenOfStr} > \text{ati}_\text{kernelDataLength}) \]
\[ \text{true} \]
\[ \text{extractQuotMarksFromStr giường text, ati}_\text{kernelDataParamBuff[3]} \]
\[ *\text{text} \triangleq 0x00 \]

\[ \text{err} \]
\[ \text{free(text)} \]
\[ \text{ifError(0x00, CME_ERR, CME_TEXT_STRING_TOO_LONG, INTERNAL_CMD_ERR)} \]
\[ \text{pbDatalpbTyp} \triangleq \text{ati}_\text{kernelDataTyp} \]
\[ \text{true} \]
\[ \text{if(atoi(strIndex) = 0) \&\& (atoi(Number) = 0)} \]
\[ \text{false} \]
\[ \text{free(text)} \]
\[ \text{ifError(0x00, CME_ERR, CME_INVALID_INDEX, INTERNAL_CMD_ERR)} \]

"if the parameter <number> is left out in the write command, the entry with the given index shall be deleted by calling servApbDelete"
Procedure writePhbookEntry

Specify phonebook entry in location number <index> in the current phonebook memory storage selected with +CPBS. Entry fields written: phone number <number> and text associated with the number.

Number := \( \text{ARX\_MAX\_NUM\_OF\_DIGITS + 1} \)

- checkPlusCharInStr((char *) ati KERNEL Data pumpedBuff[1]) = TRUE

ntError := [CME\_ERR, CME\_DIALSTRING\_TOO\_LONG, INTERNAL\_CMD\_ERR]

- type := TDA\_PLUS

<table>
<thead>
<tr>
<th>Number[0] := '4'</th>
</tr>
</thead>
<tbody>
<tr>
<td>extractQuotMarks FromStr(number, ati KERNEL Data pumpedBuff[1])</td>
</tr>
<tr>
<td>QuotMarkS removed number, ati KERNEL Data pumpedBuff[1]</td>
</tr>
</tbody>
</table>

ati_KERNEL Data pumpedTyp := PB\_FDN

- sim\_serv_valid(SIM\_SERV\_EXTA)

| lengthNbr := 20 |

| Ati_KERNEL Data pumpedTyp := PB\_FDN |

- sim\_serv_valid(SIM\_SERV\_EXTA)
procedure writePhbookEntry

// A phonebook entry in location number <index> in the current
// phonebook memory storage selected with +CPBS. Entry fields written
// include number <number> and text associated with the number.

// ati_checkPin()

False

True

if (ati_kernelDataTpTyp == PB_UNDEFINED) ||
   (ati_kernelDataTpTyp == PB_FDN)
   printError(CME_ERR, CME_OPR_NOT_ALLOWED,
              INTERNAL_CMD_ERR)

False

initialize index, type, number and text

ati_kernelDataNumOfParams > 0

(True
   (lenOfIndex = strlen(ati_kernelData
                           paramBuff[0])) > 0)
   index := atoi((char *) ati_kernelData
                  paramBuff[0])

index > ati_kernelDataTpTotal

(True
   printError(CME_ERR, CME_INVALID_INDEX,
              INTERNAL_CMD_ERR))
Procedure WriteMsgToMemory

```
procedure WriteMsgToMemory

true

mode := PDU

false

strlen(atl_paramBuff[0])

result := strlen(atl_paramBuff[0])

true

result > MAX

false

 LENGTH_OF_DA

SendErrToTE

strncpy(address, atl_paramBuff[0])

result

false

false

true

atl_numOfParams := 2

atol(atl_paramBuff[1])

stat = result

false

false

true

Conv1To1HexFrom2Hex(MsgText, Length, MsgTextBuf, MsgDataBuf)

Two IRA character is converted to one byte. (e.g., two IRA characters (IRA 50 and IRA 65) is converted to integer value 42.)

Write the address of MsgDataBuf into setVinfoPownPage

1

2

false

false

true

atl_numOfParams := 3

atol(atl_paramBuff[1])

stat = result
```
procedure writeEchoMode

generate

{&ati_data.echoMode,
ATL_SET,2,V25TER_ERR}
procedure writeCmdLineTermChar

genericFunction

(&ati_data.cmdLineTermChar,
 CARRIAGE_RETURN,128,V25TER_ERR)
procedure setTextModeParams

1

strip(atl_paramBuff[2]) = 0
false
true

profilepid := atl_param Buff[2]

strip(atl_paramBuff[3]) = 0
false
true

profile docs := atl_param Buff[3]
Procedure setOperatorSelection

1. mode = 0
   - False
     - mode = 1
       - lastExecutedCmd := SET_PLMN_CMD
         - ati_executeData!
         - ati_kernelData!
         - cmdRespType := INTERNAL_CMD_OK
2. mode = 2
   - True
     - lastExecutedCmd := SET_PLMN_CMD
3. mode = 3
   - False
   - mode = 4
   - ExecutedCmd := SET_PLMN_AUTO_CMD
     - returnValue := servPlmnSetAutomatic(PLMN_IDN_MO_OPERATION, ati_servPlmnSetAutomaticCBF)
4. stError
   - CME_ERR, CME_OPR_NOT_ALLOWED, INTERNAL_CMD_ERR
procedure setPauseBeforeBlindDial

\texttt{(\&ati\_kernelData.pauseBeforeBlindDial,}
\texttt{PAUSE\_BEFORE\_DIAL\_DEFAULT,}
\texttt{MAX\_PAUSE\_BEFORE\_DIAL\_VALUE, V25TER\_ERR)}
procedure setResultCodeSelectData

generic

begin
'/&at!_kernelData\resultCodeSelectData,'

ATI_RESET, MAX_RES_CODE_SEL_DATA_VALUE,

V25TER_ERR)

end;
procedure setOperatorSelection

1

strcpy(mncStr, atkernedata[paramBuff2][3])

lenOfStr = ?

mncStr[2] := 'F'

mncStr[3] := 0x00

strcpy(mccStr, atkernedata[paramBuff2])

mccStr[3] := 0x00

at_ejecuterData[mnc] := (Blt16) convertStrToHEX(mncStr)

at_ejecuterData[mcc] := (Blt16) convertStrToHEX(mccStr)

2

printError(CME_ERR, CME_OPR_NOT_ALLOWED, INTERNAL_CMD_ERR)

3
procedure setNetworkReg

genericFunctionOne

(&atil_kernelData|networkReg,
ATI_DISABLE, 2, CME_ERR)
procedure setAudioModeSel

returnOk()


procedure servApbReleaseCallBackFct

```c
sizeof (ati_param_apb_release_fct *)

(ati_param_apb_release_fct *)
memory allocated (result)

ati_callBackBuff := result
ati_callBackBuff -> status := status

status =

true

false

printError(CME_ERR, PHONE_ERR, INTERNAL_CMD_ERR)
```

```
subscribe (SIC_APB_CLOSED, (cafResultCb_t *) & servApbReleaseResult)

pbData)

subscribe (SIC_APB_LOADED, (cafResultCb_t *) & servApbLoadResult)
```

false

continue

false

continue

false

continue

false

continue

false

continue

false

continue
1. Write the address of MsgDataBuff into servinfopCurrPage.

2. WriteMsgBuf
da, toda, length, 
ati_executorData[msgDataBuff, 2]

servSmsSendData 
{msgRef, ati_executorData,msgDataBuff}

ReturnValue
else

return_value

prepare SMS - SUBMIT message, or
SMS - DELIVER message and write it to buffer.
{tp-mili, tp-rd, tp-vrf, tp-sri, tp-udhi,
{tp-rp, tp-da, tp-pid, tp-dcs, tp-vp,
{user data length and user data are written
to buffer. The address of buffer is written
into servinfopCurrPage.
Procedure sendMsg

mode = PDU

true

\text{_strlen(ati\_param\_Buff[0])}

false

result > MAX

true

SendErrToTE(CMS\_ERR, SL\_CMD\_ERR)

false

strcpy(da, ati\_param\_Buff[0])

false

atoi\_numOfParams = 2

true

false

\text{ConvertTo1HexFrom2Hex(length, MsgTextBuff, MsgDataBuff)}

1

\text{ConvertDataForNetwork(length, MsgTextBuff, MsgDataBuff)}

2

Two IRA character is converted to one byte. (e.g. two IRA characters (IRA 50 and IRA 65) is converted to integer value 42.)

Entered text (that is stored in MsgTextBuff) is converted according to dcs, charset in TEXT mode.

Converted data is stored into MsgDataBuff. Converted message's length is written into length.
procedure sendCmd

1. Write the address of MsgDataBuff into servinfoFpCurrPage

   servSmsSendData(msgRef, ati, executorData, msgDataBuff)

   - if ReturnValue = SERV_FAILED
     - printError(CMS_ERR, User_Data, SL_CMD_ERR)
     - exit
   - else
     - prepare SMS - COMMAND message, and write it to buffer.
     - tp-mti, tp-arr, user data length
     - and user data are written to buffer. The address of buffer is written into servinfoFpCurrPage.

2. WriteCmdToBuff(tp, ct, mn, da, to, pid, length, MsgDataBuff)
procedure selectSmsService

prepareCmdResponse

result := FALSE

false

true

atoi(ati_kernelDataParamBuff[0])

result = 0

SendErrToTE (CMS_ERR)

(CMD_END)

ati_cmdStatus :=

ati_cmdRespType := INTERNAL_CMD_OK

ati_kernelDataCmdRespTextBuff

"+CSMS: 1,1.0"

Sprintf

cmd_response := TRUE

X

X

X

X
procedure returnToOnlineState

genericFunction "we ()"
procedure readServCentreNum

\[
\text{genericFunctionThread}^{\text{\texttt{genericFunctionThread}}}(\text{\texttt{atli\_kernel\_DatalserviceCentreNum,\^VSCN: \%u}})
\]
procedure readSelPhbookMemStorage

begin

prepareCmdResponse

result = FALSE

if att_kernelData then

if pbTyp = PB_ADN then

true

else

false

end if

if pbTyp = PB_FDN then

true

else

false

end if

end if

if pbTyp_s = "FD" then

PB_FDN

else

PB_ADN

end if

Error( PHONE_ERR, INTERNAL_CMD_ERR)

if servApbMax then

(pbData)

else

att_kernelData

pbTotal = result

end if

if servApbNumEngines then

(pbData)

else

att_kernelData

pbUsed = result

end if

sprintf( att_kernelData.cmdResponseTextBuff, \"CPBS: \"%s\", %u, %u\", pbTyp_s, att_kernelData.pbUsed, att_kernelData.pbTotal)

end if

end procedure
procedure readRespFormatChar

genericFunction 1

read(&ati_kernelData->respFormatChar,"%03u")
procedure readProductSerialNumId

<table>
<thead>
<tr>
<th>dm_copy</th>
<th>(DM_PARAM_REC(EE_SECURITY), buf_Ptr, ATI_MOD_ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>result = DM_OK</td>
<td></td>
</tr>
</tbody>
</table>

intErr _ERROR \| result \| CMD_ERR \| genericFunction true \| (buf_Ptr, "%s OK")
Procedure readPhbookEntries

- If there are phonebook entries in location number range <index1>...<index2>,
  the current phonebook memory storage selected with +CPBS. If <index1> is out, only <index1> is returned.

- **ati_checkPin()**

- **ati_kernelData**
  - pbType := PB_UNDEFINDED
  - (index1 = ati_kernelData pbTotal)? (index1 = 2)
  - **printError()**
    - CME_ERR, CME_OPR_NOT_ALLOWED, INTERNAL_CMD_ERR
  - **printError()**
    - CME_ERR, CME_INVALID_INDEX, INTERNAL_CMD_ERR

- **ati_kernelData**
  - numOfParames := 2
  - False
    - index2 := index2
  - False
    - lastExecutedCmd := READ_PB_ENTRY
  - False
    - (index1 = index2) & (index2 = EMPTY)
      - **servPbsaRead(ati_kernelData, pbType, index1, FALSE, ati_servPbsaReadCallBackFct)
      - **servPbsaRead(ati_kernelData, pbType, index1, FALSE, ati_servPbsaReadCallBackFct)**

- **returnValue**
  - 1
procedure readOperatorSelection

1

(True)

sprinf

(ati_kemelDalalcmdRespTextBuff, "$COPS: %u", mode)

(cmd_response(TRUE))

mcc &< 0x0ff

mnc &< 0x0ff

\l mnc >> 4

sprinf

(ati_kemelDalalcmdRespTextBuff, "$COPS: %u,2,"\"%03x%02x\", mode, plmninfo.mcc, plmninfo.mnc)

cmd_response(TRUE)
procedure readNetworkReg

  networkReg in ati_kemelData

  servPminInfoServiceMode(&pminServiceMode)
  this function returns the service mode of the selected PLMN

  servPminInfoRoamingStatus(&pminRoamingStatus)
  this function returns the roaming status of the selected PLMN

  status := getPminStatus(pminServiceMode, pminRoamingStatus, FALSE)

  prepareCmdResponse

  sprintf(ati_kemelData.cmdResFexBuff, "%s",
          "CREG: %u,%u",
          ati_kemelData.networkReg.status)

  cmd_response(TRUE)
procedure readLocalFlowControl

genericFunction two (*IFC: 0,0*)
procedure readCurrMsgService

genericFunction two (+CSMS: 0,1,1,1)
procedure readCurrMsgFormat

genericFunctionThread

(\&ati_kernelDataIsmsInterfaceMode,"+CMGF: %u")
procedure readConnectCompTimeout

generic

end

ati_kernelDataConnectCompTimeout("%3u")
procedure readCommaDialModeTime

generic
  FunctionThree procedure (&ati_kernelData->commaDialModeTime, "%u")
procedure readCmdLineEditChar

genericFunctionTired

(&ati_data.cmdLineEditChar,"%03u")
procedure readCallingLineIdPrsn

\[\text{genericFunctionTrend}(\text{&ati\_kernel\_data\_calling\_LineIdPrsn,}^{*}+\text{CLP:} \%u)\]
procedure readAutoAnswer

```c

genericFunctionThree

(&ati_kerneldatalautoAnswer,"%3u")
```

...
procedure listSuppWritePhbookEntry

[Diagram with flowchart]

- ati_checkPin1()
- ati_kernelData & typ
  - PB_UNDEFINE
    - printError
      - (CME_ERR, CME_OPR_NOT_ALLOWED, INTERNAL_CMD_ERR)
- prepareCmdResponse
- sprintf
  - ati_kernelData.cmdRespTextBuff
    - "+CPBW: (1-%u),%u,(129,145),%u",
    - ati_kernelData.pbTotal,
    - ati_kernelData.length,
    - ati_kernelData.length
- cmd_response(TRUE)

[End of Document]
procedure listSuppSignalQuality

genericFunction

(\text{+CSQ: (2-25,99),(0,2-7,99)})
genericFunction two

"CPBS: ("FD", "SM")"
procedure listSuppRepMEErr

--- supported or status.

--- genericFunction two

--- (CMEE: 0-2)
procedure listSuppPhoneActivityStat

```
<table>
<thead>
<tr>
<th>genericFunction</th>
<th>two</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&quot;+CPAS: 0,3,4&quot;)</td>
<td></td>
</tr>
</tbody>
</table>
```

procedure listSuppCharSet

for all supported character sets.

genericFunction two

['GSM(;', 'IHE(', 'IRA', 'PCCP437', 'PCDN;', '8859-1')]
procedure listMsg

status := 0;

ati_numOfParms = 1
true

ConvertStatusToInt

Convert stat value to integer.

'Define total listed message count'

ServSmaReadData

(SERV_READ_FIRST,0)

ReturnValue

SERV_FAILED

else

reserveLibrary := TRUE

printError

(CMS_ERR, value, SL_CMIM_ERR)

X

X
null
procedure genericFunctionTwo

the functions for: +GMX
+GMR

mem_alloc

ati_kerneldata

cmdRespTextBuf
:=result

ati_kerneldata

true

print(3_mb,

"memory alloc error")

false

strncpy(
ati_kerneldata,cmdRespTextBuf,str)

ati_cmdStatus := CMD_END

ati_cmdResponseType := INTERNAL_CMD_RESPONSE

resp_response := TRUE
procedure findPhbookEntries

1

returnValue := servApbGetCurrent(pbData, ati_servApbGetCurrentFct)

SERVSTAT_OK

SERVSTAT_RUNNING

SERVSTAT_FAILED

library := TRUE

ativerData

reserveLibrary := TRUE

lastExecutedCmd := NO_CMD

printError(CME_PHONE_FAILURE, INTERNAL_CMD_ERR)

ati_kernelData

cmdStatus := CMD_END

ati_kernelData

cmdRespType := SL_CMD_OK

X

X
procedure extendedErrReport

prepareCmdResponse

result = FALSE

sprint(ati_kernelDatacmdResponseTextBuff,"+CEER: %d:%d:%d",
ati_kernelDataCall.cause,ati_kernelDataCall.ccGsmCause,
ati_kernelDataCall.mnGsmCause,

0)
Procedure convertDataForNetwork

Length Int;
TextBuff Charstring;
MsgDataBuff Charstring;

\[ dcs = dcs & DCS\_SET \]

false
\[ dcs = 0 \]
true

HexFrom2Hex

\[ r = length \times 2 \]

GSM\_DEFAULT
HEX
THE\_OTHER\_CHSETS

ConvertTo7BitFrom8Bit
(length, TextBuff, MsgDataBuff)

ConvertTo1HexFrom2Hex
(length, TextBuff, MsgDataBuff)

ConvertToGsmFromCharSet
(length, TextBuff)

ConversionTo7BitFrom8Bit
(length, TextBuff, MsgDataBuff)

\[ length := length \div 2 \]

ConvertTo7BitFrom8Bit
(length, TextBuff, MsgDataBuff)
procedure ati_servWriteFct

failed when atapp_serv_write_crnl is received from at_mknative
if cause equals to SERV_OK, the result is prepared and TE. In the other cases, error message is sent to TE.

cause = OK

true

print Response: MGW: %d

false

sprint(CmdResponseBuff)

"CMS ERROR: %d", cause)

(CmdResponseBuff, CmdEnd, CmdResponse)

(CmdResponseBuff, CmdEnd, CmdErr)
procedure ati_servSendFct

Called when smspp_serv_send_cm is received from ati_mdb

if cause equals to SERV_OK, the result is prepared and TE. In the other cases, error message is sent to TE.

```
cause = OK
true
  sprintf(CmdResponseBuff,"CMGS: %d")
  CmdResponse (CmdResponseBuff, CmdEnd, CmdResponse)

false
  sprintf(CmdResponseBuff, "CMS ERROR: %d", cause)
  CmdResponse (CmdResponseBuff, CmdEnd, CmdErr)
```
procedure ati_servPbOpen

ptr charStar
newType pbData Ref(apbPbData_type)endNewType

begin
end

squeezedCmd = SELECT PB_MEM_STORAGE
False

result = SERVSTAT_FAILED
True

printError(CME_ERR,
PHONE_ERR,INTERNAL_CMD_ERR)

ati_kernelDataTyp := pbTyp

ptr := mem_alloc(sizeof(cdPar_t))

servPbsaReadCVP(pbTyp.ptr,
sizedof(cdPar_t))= SERVSTAT_OK

Fi}

mem_free(ptr)

ati_executeData!
lastedCmd := NO_CMD

ati_kernelData

cmdStatus := CMD_END

ati_kernelData

cmdRespType := SL_CMD_OK

cmd_response(TRUE)

ati_kernelData

ati_kernelDataLength := ptrMax_data_len

ati_kernelDataLength += 2

ati_kernelDataLength = (ati_kernelDataLength - 2) * 2

ati_kernelData

ati_kernelDataLength := ptrMax_tag_len

pbDataTyp := pbTyp

ati_kernelData

ati_kernelData

ati_kernelData

ati_kernelData

ati_kernelData
procedure ati_servApbGetCurr

```
sprintf((char *) ati_kernelData.cmdRespTextBuff, "CPBF: %d, \n\n", apbPtrApbEntryData.location, apbPtrApbEntryData.data, type, apbPtrGsm)

free(apbPtrGsm)

ati_kernelData.cmdRespType := SL_CMD_RESPONSE

cmd_response(TRUE)
```
procedure ati_basicSpeechCall

\[
\begin{align*}
\text{ati_kernelDataCall} & \quad \text{Answer} = 0 \\
\text{ati_kernelDataCall} & \quad \text{Answer} = \text{ati_kernelDataCall(ch)} \\
\text{return_value} & \quad \text{printError(V25TER_ERR, NO_CARRIER_VALUE, SL_CMD_ERR)} \\
\end{align*}
\]
dure ati_basicSpeechCall

4

ati_kernalDataCall:
  ton := ton

return_value := servCallInfo

INFO_NAME_BY_CH, ati_kernalDataCall
  call.ch.name

return_value := ATI_TRUE

name[0] != EMPTY

true

store name in ati_kernalDataCall.name

false

send CLIP

5
procedure ati_basicSpeechCall

1

indication = CSE_END_IND

True

False

CallHandle

Indication = CSE_FINISH_CONF_IND

False

CmdResponse = (OKEY, CmdEnd, CmdOk)

return_value = servCallFree(at_kernelDataCall.ch)

saveCauseValue

This function writes the call release causes in ati_kernelData for later interrogation...

return_value = TRUE

True

at_kernelDataCall.callState

false

CmdResponse = (OKEY, CmdEnd, CmdOk)

CallHandle

Indication = CSE_FINISH_CONF_IND

True

CallHandle

Indication = CSE_FINISH_CONF_IND

true

stop ringing

false

Result: Code Selection(X) is checked.

CmdResponses

at_kernelDataCall.state := CALL_IDLE

(OKEY, CmdEnd, CmdOk)
EXECUTER

The EXECUTER has three states: idle, WaitForCmd, and WaitForCnf.

- If ForCmd state, EXECUTER waits for a command signal from the PARSER.
- If ForCnf state, EXECUTER waits for a command signal from the Service Library.
- If Cnf Cmd is same as ATI_EIDLE, WaitForCnf is same as ATI_ecnfWait.

- if (ForCnf)
  - servPlmnRequestSelectCBF
  - nit_servPlmnSetAutomaticCBF
  - nit_servPlmnRequestListCBF

- nRequestSelect
  - nit_servPlmnSetAutomatic
  - nit_servPlmnRequestList

- if (ForCmd)
  - WaitForCmd
  - WaitForCmd
ss EXECUTER

The EXECUTER has three states: idle, WaitForCmd, and WaitForCnf. In the idle state, executer waits for command signal from the PARSER. When the PARSER decides the command, it sends cmd signal to EXECUTER. (idle state, parser calls the related function from executor). If the cmnd, EXECUTER waits for call back signal from Service Library. (cmd, call back functions are waited). If cmnd is same as ATI_EIDLE, WaitForCnf is same as ATI_ECNPWAIT.

Diagram:

1. tForCnf
   - servAppGetCurrCallBackFct
   - ApbGetCurr
   - servApbReleaseCallBackFct
   - WaitForCmd
   - servSecPskPukVerifyCallBackFct
   - WaitForCmd
   - ati_servSsSendRequestCBF
   - WaitForCmd
The executor has three states: idle, WaitForCmd, and WaitForCnf.

- Idle: The executor waits for a command signal from the parser.
- WaitForCmd: The executor waits for a command, then sends a cmd signal to the executor.
- WaitForCnf: The executor waits for a callback signal from the service library.

ForCmd state: The executor waits for a command signal from the parser.
ForCnf state: The executor waits for a callback signal from the service library.

The diagram shows the flow of control between these states and functions, including:
- `servReadDataCallBackFct`: Reading data callback function.
- `ati_servWriteDataCallBackFct`: Writing data callback function.
- `ati_servSendDataCallBackFct`: Sending data callback function.
- `ati_servDeleteCallbackFct`: Deleting callback function.

The flow includes decisions based on conditions like 'true' and 'false', transitioning between states and functions accordingly.
The EXECUTER class has three states: Idle, WaitForCmd, and WaitForCnf.

- In the Idle state, the EXECUTER waits for a command signal from the PARSER.
- In the WaitForCmd state, the EXECUTER waits for a command signal from the PARSER.
- In the WaitForCnf state, the EXECUTER waits for a callback from the Service Library.

A callback is a function that is called back when a command is completed.

The diagram shows the flow of control between these states and the conditions under which the states transition.
EXECUTER

The EXECUTER has three states. They are idle, WaitForCmd, and WaitForCnf.

- If the state is Idle, the EXECUTER waits for a command signal from the PARSER.
- If the state is WaitForCmd, the EXECUTER waits for a command signal to EXECUTER. When the command is received, the EXECUTER calls the related function from the executor.
- If the state is WaitForCnf, the EXECUTER waits for a confirmation signal from the Service Library.

Command (Cmd) and confirmation (Cnf) signals are handled as follows:
- If the command is the same as ATL_EIDLE, the EXECUTER switches to the Idle state.
- If the command is the same as ATL_ECNPWAIT, the EXECUTER waits for a confirmation signal.
- If the command is different from the above, the EXECUTER processes the command as follows:
  - If the command is to enter a pin (enterPinCmd), the EXECUTER reads the entered pin (readEnterPin) and then waits for a confirmation signal (true)
  - If the command is to charge the battery (batteryChargeCmd), the EXECUTER charges the battery (batteryCharge) and then waits for a confirmation signal (false)
  - If the command is to list supported battery charges (listSuppBatteryChargeCmd), the EXECUTER lists the supported battery charges (listSuppBatteryCharge) and then waits for a confirmation signal (false)
EXECUTER

Executing has three states. They are Idle, WaitForCmd, and WaitForCnf.
Forward state, executor waits for command signal from the PARSER.
Parser decides the command, sends cmd signal to EXECUTER.
EXECUTER waits for callback from Service Library.
Parser calls the related function from executor.
Forward, EXECUTER waits for callback from Service Library.
Parser calls the related function from executor.

ForCmd

AudioModeSelCmd
readTerminalIdCmd
setSelCellBroadMsgTypeCmd
setCallingLineIdPrsnCmd

dioModeSel
readTerminalId
setSelCellBroadMsgType
setCallingLineIdPrsn
EXECUTER

ForCmd state, the executor waits command signal from the PARSER. PARSER decodes the command, it sends cmd signal to EXECUTER, (parse parser call the related function from executor). ForCmd, EXECUTER waits call back signal from Service Library. (call back functions are waited). ForCmd is same as ATI_EIDLE, WaitForCnf is same as ATI_ECNFWAIT.

ForCmd

CommandDialModeTimeCmd

DialModeTime

readCommandDialModeTimeCmd

setAutoDisconnDelayCmd

readAutoDisconnDelayCmd

-
The executor has three states. They are idle, WaitForCmd, or WaitForCnf.

- ForCmd state, the executor waits for a command signal from the PARSER.
- ARSER decides the command, it sends cmd signal to EXECUTER.
- PDE, parser calls the related function from executor.
- ForCnf, EXECUTER waits call back signal from Service Library.
- PDE, call back functions are waited.
-_Cmd is same as ATLEIDLE, WaitForCnf is same as ATLECNFWAIT.
EXECUTER

has three states: Idle, WaitForCmd, and WaitForCnf.

- In the Idle state, the executor waits for a command signal from the PARSER.
- In the WaitForCmd state, the executor waits for a command signal and sends it to the EXECUTER.
- In the WaitForCnf state, the executor waits for a callback signal from the Service Library.

If the command is the same as the one being waited for, the executor moves to the Idle state.

The states and transitions are shown in the diagram as follows:

- `forCmd` state
  - `setTypeOfAddrCmd` transition
    - `listSuppSetTypeOfAddrCmd` transition
      - `setResultCodeCmd` transition

- `setTypeOfAddrCmd` state
  - `listSuppSetTypeOfAddrCmd` transition
    - `setResultCodeCmd` transition

- `listSuppSetTypeOfAddrCmd` state
  - `setResultCodeCmd` transition
The class EXECUTER has three states: Idle, WaitForCmd, and WaitForCnf.

In the Idle state, EXECUTER waits for a command signal from the Parser.

- If the command is WaitForCmd state, the Executer waits for a command signal from the Parser.
- If the command is WaitForCnf state, the Executer waits for a callback signal from the Service Library.

The state WaitForCmd is the same as ATL_EIDLE, and WaitForCnf is the same as ATL_ECNFWAIT.
Class EXECUTER

The class has three states. They are idle, WaitForCmd, WaitForCnf.

- If the current state is WaitForCmd, the executor waits for a command signal from the PARSER.
- If the PARSER decides to send a cmd signal to EXECUTER, the executor calls the related function from the Service Library.
- If the PARSER decides to wait for a cmd signal from Service Library, the executor waits for a cmd signal from Service Library.

- If the current state is WaitForCnf, the executor waits for a cmd signal from Service Library.

- If the current state is same as ATI_EIDLE, the executor waits for a cmd signal from ATI_ECNFWAIT.

The diagram shows the flow of execution based on the states and commands.
EXECUTER

EXECUTER has three states: idle, WaitForCmd, and WaitForCnf.
- In the WaitForCmd state, EXECUTER waits for a command signal from the PARSER.
- When the PARSER decides to execute, it sends a cmd signal to EXECUTER.
- In the WaitForCnf state, EXECUTER waits for a call back signal from the Service Library.
- The cmd and call back signals are waited for in the same state, ATL_EIDLE.

The diagram shows the state transitions and functions associated with each state.
EXECUTER has three states: idle, WaitForCmd, WaitForCnt.

- If in the WaitForCmd state, the executor waits for a command signal from the PARSER. The PARSER decides the command and sends it to the EXECUTER.
- If in the WaitForCnt state, the executor waits for a callback signal from the Service Library.
- If the command is as AT_IDLE, WaitForCnt is as AT_IDNFWAIT.

Diagram:

- CurrChar
  - listSuppChar
    - Service
      - selectSms
        - Service
          - readCurrMsg
            - Service
              - listSuppMsg
                - Storage
                  - setPrefMsg
                    - Storage
                      - readCurrPref
                        - MsgStorage

Class EXECUTER

- EXECUTER has three states: idle, WaitForCmd, WaitForCnf.
- If in WaitForCmd state, EXECUTER waits command signal from the PARSER. PARSER decides the command, sends cmd signal to EXECUTER.
- If in WaitForCnf state, EXECUTER waits command signal from Service Library. (Non-back functions are waited).
- ATL EABLE, WaitForCnf is same as ATLECNFWAIT.

- i_svc: ImnSetManual
- i_svc: ImnRequestSelect
- i_svc: ImnService
- i_svc: ImnRequestAbort
- i_svc: PboOpen
- servSec: PukVerifyCallBackFct
class ATILLA

A has two states: ATI_SDOWN, ATI_SUP. At the beginning, ATILLA's state is ATI_SDOWN.

Deallocates the necessary memory which is being used by ByteIO.

Called when a SCI_DA_IND is received with result value other than SCI_TX_BUFF_EMPTY meaning some data is ready in the SCI input data buffer to be read by ATI.

Write the byte to SCI if echo mode is set

Converts the characters in the given buffer to the uppercase letters if needed

Finds the parity mode used by remote in the command line

Clears the last bit of len bytes from the given buffer

returns byteio to command state from online data state

Allocates the necessary memory which will be used by ByteIO

sets 30 sec. timeout

reset 30 sec. timeout
A has two states: ATI_SDOWN, ATI_SUP. At the beginning, ATIILLA's state is ATI_SDOWN. When it receives ATI_START_REQ, its state goes to ATI_SUP (start up).
class ATILLA

a has two state: ATI_SDOWN, ATI_SUP. At the beginning, ATILLA's state is ATI_SDOWN. When ATI_START_REQ arrives ATI_START_REQ, its state goes to ATI_SUP (start up).

ATI_SUP

SCI_DA_IND (reason)

reason

L_TX_BUF_EMPTY

SCIOutLen := sci_ati_tx_buffers (ati_byteIO->uart)

len := ati_readFromOutQueue(ptr)

((len = 0) & (ati_dataIcmdStat = TRUE) & OutLen = MAX_RESPONSE_OUTPUT_LEN))

ati_dataIcmdStat := FALSE
ati_dataIIOState := ATI_BCMDPREFIX

len = 0

false

true

SCIWrite(ati_byteIO->uart, SCI_AT_HAYES, ptr, len)
A has two states: ATI_SDOWN, ATI_SUP. At the beginning, ATILLA's state is ATI_SDOWN. When ATILLA receives ATI_START_REQ, its state goes to ATI_SUP (start up).
ATILLA has two states: ATI_SDOWN, ATI_SUP. At the beginning, ATILLA's state is ATI_SDOWN. When ATI_START_REQ arrives, its state goes to ATI_SUP (start up).

- ATI_SUP
  - ATI_PEST.CNF
    - ifResponse
      - SUCCE
      - else
        - ati_dataprotState := ATI_PIDLE

- ATI_PREQ.CNF
  - ati_dataprotState := ATI_PIDLE
    - true
      - ati_dataprotState := ATI_PREQ
    - false
      - DEVO_PSTAT.IND
        - at
          - ACT
            - ati_dataprotState := ATI_PIDLE
          - NACT
            - AT_PEST_REQ
procedure ati_byteInputControl

if when a SCI_DA_IND is received with result value other than
TX_BUFF_EMPTY meaning some data is ready in the SCI input dis
atchOrg.chFirst Octet;

buffer buffPtr CharStar;

length, actualLength Integer

length :=
sci_rx_bufstat(ati_byteIO->uart)

length = 0

true

false

actualLength := length
buff := mem_alloc(length)
buffPtr := buff

clipboard this area.

memory allocation is done for buff.

sci_b_read(ati_byteIO->uart,SCI_AT_HAYES,buffPtr,length)

loop

loop until all bytes which are received are processed
procedure ati_bytelInputControl

3 when a SCL BA_IND is received with result value other than TX_BUFF_EMPTY meaning some data is ready in the SCI input data to be read by ATI.

1

ati_data1OState := ATI_BONLINEDATA

true

false

ati_data1OState

ATL_BCMDPREFIX

chOrg := ch 'clear most sign. bit of ch'

1

ati_bytelOatil_bytelOInputbuffPtr

3

ch := 'A' && (ch := 'a')

true

'ch is copied ati_bytelOatil_bytelOInputbuff' false

'30 sec character timeout is started'

2

4

3
procedure ati_byteInputControl

when a SCI_DA_IND is received with result value other than
TX_BUFF_EMPTY meaning some data is ready in the SCI input data
buffer to be read by ATI.

5

4

ati_dataBioState

ati_BCMDBODY

ati_BONLINEDATA

12

this is need for +CMGS,+CMGW,+CMGC
commands to receive text.

8

CARRiAGE_RETURN

ch

10

CTRL_Z

II ESC

ch

11

ati_dataCmdLineEditChar

else

else

13

ati_dataCmdLineEditChar

ati_dataCmdLineTermChar

14

timer for 0 sec.

chOrg := ch = ch & ATI_BIT8_CLEAR_LAST_BIT_MASK

ati_bytelOati_bytelOInputBuffPtr

ati_bytelOati_bytelOInputBuffLen

true

ati_findExistingParity

6

false

"chOrg is copied to
ati_bytelOati_bytelOInputBuff"

7
When a SCI_DA_IND is received with result value other than X_BUF.EMPTY meaning some data is ready in the SCI input data to be read by ATI.

7

ati_findExistingParity()

ati_clearLastBit
{ati_byteOlati_byteOInputbuff,
ati_byteOlati_byteOInputbuffPtr,
ati_byteOCharFormat}

ati_convertCmdToUpperCase
{ati_byteOlati_byteOInputbuff,
ati_byteOlati_byteOInputbuffPtr}

ati_dataBlockState
:= ATI_BCMDWAIT

CmdLineInd
{ati-byteOlati-byteOInputbuff,
ati-byteOlati-byteOInputbuffPtr, 2}

8

'check last character, it can be only CTRL-Z or ESC, ch is copied to buff'

ati_clearLastBit
{ati-byteOlati-byteOInputbuff,
ati-byteOlati-byteOInputbuffPtr,
ati-byteOCharFormat}

call blockState :=
ATI_BMSGTXTWAIT

CmdLineInd
{ati-byteOlati-byteOInputbuff,
ati-byteOlati-byteOInputbuffPtr}
procedure ATILLA-> ati_clearLastBit

T buff CharStar,
T len Octet,
T format ati_charFormatType;

i Octet;

(format = ATI_CF_NO)

true

''clear last bit in buff''

=format = ATI_CF_BN4)
procedure ati_deallocateByteIOInputMem

allocates the necessary memory which is being used by...

mem_free
(ati_byteIOlati_byteIOInputbuff)


procedure ati_emptyOutQue

returnValue Octet;

true
ati_deleteFirstEntryFromOutQue()

returnValue
false
procedure ati_resetTimer

sec. timeout for functionality

X
procedure connComplTimeout

- servCallEndAll()

allKernelDatacall
connComplTimeout := TRUE

X
dure ringTimeout

DCL
returnValue Octet

aten Cellular

state := WAITING_CALL_IN_PROGRESS

True

setTimer(atiTingTimer)

deDataCallCounter := 0

send RING or CRING according to ati_kernelData!
cellularResultCode

*send CLIP if ati_kernelData!
callingLineIdPres data is ENABLE

ati_setTimer(at_data, atiTingTimer,
ATI_RING_EXPIRY_IND,
ATI_SEC_TIMEOUT_VALUE)

ati_kernelDataCall!
ringCounter++

ati_kernelData!

dataAnswer := 0

ati_kernelData!autoAnswer l==
ati_kernelDataCall!ringCounter

True

1
ex EXECUTER

exer has three state. They are idle, WaitForCmd, WaitForCnt.

if ForCnt state,excuter waits command signal from the PARSER.
PARSER decide the command, it sends cmd signal to EXECUTER.
ExeCore, parser calls the related function from executer).

if ForCnt, EXECUTER waits call back signal from Service Library.
ExeCore,call back functions are waited).

ExeCore is same as ATI_EIDLE, WaitForCnt is same as ATI_ECNFWAIT;

TYPE ati_paramBuffLength Int = 6;
TYPE ati_paramBuffType
NEWTYPE ati_paramBuffType;

/**************************** DECLARATIONS ******************************/

.s3Value,s4Value,s5Value,echoValue,qValue,vValue,chset,service,mode,show,cmdValue Integer;
.srvInfosrvInfoType;
.msiIndParam MsgIndParamType;
.profile ParamSetType;
.returnValue,resize,library ReturnType;
.cause CauseType;
.primitiveHead PrimitiveHeadType;
.MsgTextBuff Charstring;
.length Integer;
.ati_paramBuff ati_paramBuffType;
.ati_numOfParams Integer;
.CmdResponseBuff Charstring;
.result Integer;

/**************************** DECLARATIONS ******************************/
The EXECUTOR has three states. They are Idle, WaitForCmd, WaitForStat.

- Idle: If the EXECUTOR state is Idle, the EXECUTOR waits for the command signal from the PARSER. If the command is 'at', the EXECUTOR issues a 'wait' signal to the PARSER.
- WaitForCmd: If the EXECUTOR state is WaitForCmd, it sends the cmd signal to the PARSER.
- WaitForStat: If the EXECUTOR state is WaitForStat, it sends the cmd signal to the PARSER.

The EXECUTOR waits for a command signal from the Service Library.

- type ati_exectutorData_t
  - address CharStar;
  - gDataBuff CharStar;
  - gAddress Octet;
  - gStatus Octet;
  - gCount Count Octet;
  - nExecutedCmd Octet;
  - nListedMsgCount Octet;
  - gNumber Character;
  - ic Integer;
  - xc Integer;

- type smsMsgType_t
  - ERALS
  - TL_SMS_DELIVER,
  - TL_SMS_SUBMIT,
  - TL_SMS_STATUS;

- typedef msmStatusType_t
  - ERALS
  - EC_UNREAD,
  - EC_READ,
  - TO_UNSSENT,
  - TO_SENT,
  - UM_OF_MSG_STATUS;

- typedef executedCmdType_t
  - ERALS
  - O_CMD,
  - END_MSG,
  - ST_MSG,
  - END_MSG_FROM_MEM,
  - END_CMD,
  - RITE_MSG_TO_MEM,
  - ELECT_PB_MEM_STORAGE,
  - EAD_PB_ENTRY,
  - RITE PB_ENTRY,
  - IND_PB_ENTRY,
  - ET_ENTER_PIN,
  - OCK CONTROL,
  - ST_PLMN,
  - ET PLMN CMD,
  - ET PLMN_AUTO_CMD.

The type ati_exectutorData_t declaration includes fields for handling command signals, addresses, and status information.

The type smsMsgType_t defines the types of SMS messages that can be handled by the EXECUTOR.

The type executedCmdType_t defines the types of commands that can be executed by the EXECUTOR.
# EXECUTER

It has three states: they are idle, WaitForCmd, WaitForCNF.

ForCmd state, executor waits command signal from the PARSER.
PARSER decides the command; it sends cmd signal to EXECUTER.
(idc, parser calls the related function from executor). 
(ForCNF, EXECUTER waits call back signal from Service Library. 
code, call back functions are waited).

ForCmd is same as ATI_EIDLE, WaitForCmd is same as ATI_ECNPWAIT.

```c
int SERVSTAT_FALSE = 0
int SERVSTAT_TRUE = 1

int DUMMY = 0 /* PageTags */
int READ = 1
int NOT_READ = 3
int SENT = 5
int NOT_SENT = 7

int NUM_OF_STATUS = 5
int NUM_OF_PB_TYPE = 6

int SM_STR_LEN = 5
int MAX_DA_LEN = 24
int MAX_SCA_LEN = 24
int MAX_ADDRESS_LEN = 24
int MAX_CONV_BUFF_LEN = 24
int MAX_CONV_SCTS_LEN = 24
int MAX_VP_STR_LEN = 22
int SCTS_LEN = 7
int VP_LEN = 7
int DT_LEN = 7

int MTL_SET = 0x03
int SMS_SET = 0x04
int SEND_SET = 0x20
int RP_SET = 0x80
int RD_SET = 0x04
int VPF_SET = 0x18
int SRR_SET = 0x20

int SHOW_HEADER = 0x01
```

- `MAX_STATUS_INFO_LEN = 13`
- `MAX_CHSET_STR_LEN = 8`
- `MAX_TEXT_MSG_LEN = 480`
- `MAX_PDU_MSG_LEN = 420`
- `MAX_CMD_TEXT_LEN = 380`
- `MAX_PDU_STATUS_MSG_LEN = 100`
- `MAX_TEXT_STATUS_MSG_LEN = 100`
EXECUTER

EXECUTER has three states. They are idle, WaitForCmd, WaitForCnf.

- **WaitForCmd** state, executor waits command signal from the PARSER.
- **WaitForCnf** state, executor waits command signal from EXECUTER.
- **PARSER** decides the command, it sends cmd signal to EXECUTER.
- **EXECUTER** waits for the related function from executor.
- **PARSER** decides the command, it sends cmd signal to EXECUTER.
- **EXECUTER** waits for the related function from executor.
- **PARSER** decides the command, it sends cmd signal to EXECUTER.
- **EXECUTER** waits for the related function from executor.

**writeCmd**

**writeRespFormatChar**

**writeCmd**

**writeLineEditChar**

**writeEchoMode**

**writeResultCodeSuppMode**

**writeDcoRespFormat**

**setToDefaultConfig**

**setToFactoryConfig**

**addManufacturerId**

**addModelId**

**addRevisionId**

**addCompleteCapab**
EXECUTER waits call back signal from Service Library.
back functions are waited).

Attempt: WRITE, WaitForCnt is same as ATL_CNF_WAIT.

SetTextModeParams

SetTextModeParams

setLocalFlowControl

writeResCodeHeadPrsnParams

readResCodeHeadPrsnParams

setSuppResCodeHeadPrsnParams

saveSettings

setSuppMsSettingProfileNo

restoreSettings

setSuppProfileNo

writeMsgRcvIndProc

readCurMsgRcvIndProc
EXECUTER has three states. They are idle, WaitForCmd, WaitForCnf.

- When in the WaitForCmd state, EXECUTER waits for a command signal from the PARSER.
- When in the WaitForCnf state, EXECUTER waits for a confirmation signal from the PARSER.
- When in the idle state, EXECUTER is ready to receive commands.

Back functions are waited for.

Example: If the PARSER is in the idle state, and the PARSER waits a command, the PARSER calls the related function from the EXECUTER.

- If the PARSER is in the WaitForCmd state, the PARSER waits for a confirmation signal from the PARSER.
- If the PARSER is in the WaitForCnf state, the PARSER waits for a confirmation signal from the PARSER.
- If the PARSER is in the idle state, the PARSER is ready to receive commands.

Example: If the PARSER is in the idle state, and the PARSER waits a confirmation signal, the PARSER calls the related function from the EXECUTER.

Example: If the PARSER is in the WaitForCmd state, the PARSER waits for a confirmation signal from the PARSER.

Example: If the PARSER is in the WaitForCnf state, the PARSER waits for a confirmation signal from the PARSER.

Example: If the PARSER is in the idle state, the PARSER is ready to receive commands.

Example: If the PARSER is in the idle state, and the PARSER waits a command, the PARSER calls the related function from the EXECUTER.

Example: If the PARSER is in the WaitForCmd state, the PARSER waits for a confirmation signal from the PARSER.

Example: If the PARSER is in the WaitForCnf state, the PARSER waits for a confirmation signal from the PARSER.

Example: If the PARSER is in the idle state, the PARSER is ready to receive commands.

Example: If the PARSER is in the idle state, and the PARSER waits a confirmation signal, the PARSER calls the related function from the EXECUTER.

Example: If the PARSER is in the WaitForCmd state, the PARSER waits for a confirmation signal from the PARSER.

Example: If the PARSER is in the WaitForCnf state, the PARSER waits for a confirmation signal from the PARSER.

Example: If the PARSER is in the idle state, the PARSER is ready to receive commands.
EXECUTER has three states: Idle, WaitForCmd, and WaitForCnf.
- In Idle state, EXECUTER waits for a command signal from the PARSER.
- In WaitForCmd state, EXECUTER waits for a command signal from the PARSER.
- In WaitForCnf state, EXECUTER waits for a command signal from the Service Library.

When an event occurs, EXECUTER transitions to the Idle state.
EXECUTER

EXECUTER has three states: they are Idle, WaitForCmd, WaitForCnf.

- Idle: EXECUTER waits command signal from the PARSER. PARSER decides the command, it sends cmd signal to EXECUTER.
- WaitForCmd: EXECUTER waits for a callback signal from Service Library. (back functions are waited).
- WaitForCnf: EXECUTER waits callback signal from Service Library (in ATLEIDLE, WaitForCnf is same as ATLECNFWAIT).

- GetPhbookEntries
- WritePhbookEntry
- GetSuppWritePhbookEntry
- GetSelTypeOfAddr
- setSelectedTypeOfAddr
- GetSuppSelTypeOfAddr
- GetResultCell
- GetCellResultCode
- GetCellResultCode
- GetSuppCellResultCode
- GetErrReport
- GetNetworkReg
- GetNetworkReg
SYN TYPE C m d L in E l n T y p e = I n t
   C O N S T A N T S 0:80
   E N D S Y N T Y P E ;

SYN TYPE RespLineLenType = Int
   C O N S T A N T S 0:400
   E N D S Y N T Y P E ;

N E W T Y P E ByteInterfa ce M o de T y p e
   L I T E R A L S  C m d L in e M o de , W a it M o de , C o n t i n u e M o de , T e x t M o de
   E N D N E W T Y P E ByteI nterfaceM o de T y p e ;

S I G N A L
   S o f t R e s e t ,
   C m d L in e I n d ( C h a r s t r i n g , C m d L in e L e n T y p e ) ,
   R e s p o n s e I n d ( C h a r s t r i n g , B y t e I n t e r f a c e M o de T y p e , C m d L in e L e n T y p e , R e s p L i n e L e n T y p e ) ,
   S e t E c h o M o d e ( E c h o M o de T y p e ) ,
   S e t C m d L in e T e r m ( C h a r a c t e r ) ,
   S e t C m d L in e E d i t ( C h a r a c t e r ) ;
e smacModeType
ALS
/CB_SUBSCRIBE,
/CB_UNSUBSCRIBE,
/CB_RETRIGGER,
/CBMODE_T_MAX
type smacModeType;

: smacCauseType = Integer endsnynptype;
:
: apbTriggersType
ALS
/STARTUP,
/SHUTDOWN,
DELETEALL_START,
DELETEALL_END,
DELETEALL_FAILED
type apbTriggersType;

: pbsaTriggersType
ALS
T_OK = SERVSTAT_OK,
T_FAILED = SERVSTAT_FAILED,
T_EMPTY,
T_PURGING,
T_READ,
T_STARTUP,
T_SHUTDOWN
type pbsaTriggersType;

: struct pbsaEntryDataType
  e  pbsaPbType;
  n Octet;
  catStatusType;
  e alpha Array(MAX_NAME_LENGTH + 1, Character) endnynptype;
  e data Array(MAX_NUMBER_LENGTH + 1, Character) endnynptype;
type pbsaEntryDataType;

: struct apbEntryDataType
  e  apbPbType;
  n Octet;
  catStatusType;
  e  alpha Array(MAX_NAME_LENGTH + 1, Character) endnynptype;
  e  data Array(MAX_NUMBER_LENGTH + 1, Character) endnynptype;
ype apbEntryDatatype;

apbEntryDataType Star (apbEntryDataType) endnynptype;
pbsaEntryData Ref (pbsaEntryDataType) endnynptype;
typedef apbPbtye = Octet endsyntype;
typedef apbLocType = Octet endsyntype;

typenew apbPbData struct
    ptyp apbPbType /* phonebook type */
    loc apbLocType /* current location */
endernewtype;

typenew apbPbDataStar Ref (apbPbdata) endnewtype;

typenew pbsaPbType = Octet endsyntype;

typenew pbsaLocType = Octet endsyntype;

typenew mobstatBatteryLevel
    LITERALS BLEV_FULL,BLEV_3,BLEV_2,BLEV_1,BLEV_EMPTY
endernewtype mobstatBatteryLevel;

typenew plmnIdNumType
    LITERALS PLMN_IDN_MT_OPERATION,/* 0 = mobile terminated operation */
        PLMN_IDN_MO_OPERATION,/* 1 = mobile originated operation */
        MAX_PLMN_IDN
endernewtype plmnIdNumType;

typenew plmnServiceMode = Integer endsyntype;

typenew plmnServiceModeStar Ref (plmnServiceMode) endnewtype;

typenew struct plmnSignalType
    snr Integer,
    ndev Integer,
    nbars Octet
derndernewtype plmnSignalType;

typenew plmnSignalStar Ref (plmnSignalType) endnewtype;

typenew plmnRoamingStatus
    LITERALS
        PLMN_RS_UNDEF,
        PLMN_RS_AT_HOME,
        PLMN_RS_NATIONAL,
        PLMN_RS_INTERNATIONAL,
        MAX_PLMN_RS
endernewtype plmnRoamingStatus_t;

typenew plmnInfoRoamingStar Ref (plmnInfoRoamingType) endnewtype;
NEWTYPE TrigModeType
LITERALS ServTrigger, ServNoTrigger
ENDNEWTYPE TrigModeType

NEWTYPE DevcoStateType
LITERALS Devco_Act, Devco_Nact;
ENDNEWTYPE DevcoStateType;

NEWTYPE DevcoResultType
LITERALS Devco_Succ, Devco_Fail;
ENDNEWTYPE DevcoResultType;

NEWTYPE EchoModeType
LITERALS ECHOSET, ECHORESET;
ENDNEWTYPE EchoModeType;

SYNTYPE InLenType = Integer
ENDSYNTYPE;

SYNTYPE OutLenType = Integer
ENDSYNTYPE;

NEWTYPE SciUserType
LITERALS SCI_NO_UART,
SCI_1ST_UART,
SCI_2ND_UART,
SCI_3RD_UART,
SCI_4TH_UART,
SCI_MAX_UART;
ENDNEWTYPE SciUserType;

NEWTYPE SciPathType
LITERALS SCI_NO_PATH, /* Only used for active paths */
SCI_SLP_RCH,
SCI_SLP_SCH,
SCI_SLP_MCH_1,
SCI_SLP_MCH_0,
SCI_SAS, /* Not supported in ARM-based designs */
SCI_RTC,
SCI_CONSOLE,
SCI_AT_HAYES,
SCI_IRDA,
SCI_MAX_PATH;
ENDNEWTYPE SciPathType;

NEWTYPE SciMailReasonType
LITERALS SCI_NO_MAIL_REASON, /* No reason for sending a mail */
SCI_RX_BUF_EMPTY, /* Rx buffer is empty */
SCI_RX_BUF_TIMEOUT, /* No change in the Rx buffer for a certain time */
SCI_RX_BUF_FILLED, /* Rx buffer contents above SCI's trigger level */
SCI_RX_BUF_FULL, /* Rx buffer is nearly full, danger of data loss */
SCI_RX_BUF_OVFL, /* Rx buffer overflow, data have been lost */
SCI_TX_BUF_EMPTY, /* Tx buffer is empty */
SCI_MAX_MAIL_REASON
ENDNEWTYPE SciMailReasonType;
SYNTYPE Int = Integer
ENDSYNTYPE;

SYNTYPE ResultType = Int
ENDSYNTYPE;

SYNTYPE RpMode = Int
    CONSTANTS 0:10
ENDSYNTYPE;

SYNTYPE PageNumber = Int
    CONSTANTS 0:255
ENDSYNTYPE;

SYNTYPE MsgNumber = Int
    CONSTANTS 0:255
ENDSYNTYPE;

NEWTYPE ServModeType
    LITERALS ServIndSm, ServReqSm;
ENDNEWTYPE ServModeType;

NEWTYPE UpdModeType
    LITERALS ServAuto, ServManual;
ENDNEWTYPE UpdModeType;

NEWTYPE LocModeType
    LITERALS ServFree, ServSameRead, ServSameWrite, ServAbsent;
ENDNEWTYPE LocModeType;

NEWTYPE ReadModeType
    LITERALS ServReadFirst, ServReadNext, ServReadAbsolute;
ENDNEWTYPE ReadModeType;

NEWTYPE CategoryType
    LITERALS ServDummy, ServUnread, ServRead, ServUnsent, Serv;
ENDNEWTYPE CategoryType;

NEWTYPE DellModeType
    LITERALS ServCurrent, ServNumber, ServAll;
ENDNEWTYPE DellModeType;

NEWTYPE CauseType
    LITERALS ServOk, ServFailed, ServBusy, ServReject, ServCommFault;
    ServCanNotReadMessage, ServNotSubscribed;
ENDNEWTYPE CauseType;

NEWTYPE DevcoDstateType
    LITERALS Devco_Dact, Devco_Dnact;
ENDNEWTYPE DevcoDstateType;
/* PLMN Service Interactions */
/* ----------------------- */

  servPlmnInfoSelectionMode(plmnSelectionModeStar),
  servPlmnInfoSelectedPlmn(plmnInfoStar),
  servPlmnSetAutomatic(plmnldNumType,calResultCbStar),
  servPlmnRequestReselect(plmnldNumType,calResultCbStar),
  servPlmnSetManual(plmnldNumType,calResultCbStar),,
  servPlmnRequestSelect(plmnInfoStar,plmnldNumType,calResultCbStar),
  servPlmnAbortSelection(plmnldNumType,calResultCbStar),
  servPlmnRequestList(plmnldNumType,calResultCbStar),
  servPlmnInfoServiceMode(plmnServiceModeStar),
  servPlmnInfoSignal(plmnSignalStar),
  servPlmnInfoRoamingStatus(plmnInfoRoamingStar),

/* SMSCB Service Interactions */
/* -------------------------- */

  servSmscbSetRouting(Octet),
  servSmscbMid(Integer,smscbModeType,Octet,CharStar),
  servSmscbDcs(Integer,smscbModeType,Octet,CharStar),
  servSmscbMidUnsubscribeAll(Integer),
  servSmscbDcsUnsubscribeAll(Integer),

/* Indications */
/* ------------- */

  ati_servMsgInd(smsCauseType,CharStar),
  ati_servMsgInd(smsCauseType,CharStar),
  ati_servSmscbMsg(smscbCauseType,smscbPageStar),
  ati_basicSpeechCall(callTriggersType,voidStar),
  ati_finishCall(callTriggersType,voidStar),
  ati_servPbOpen(callStatusType,pbaaPbStar),
  ati_servPlmnService(plmnldNumType,plmnResultStar)
**SIGNAL**

" DEVCO Interactions */
"-------------------- */
ATI_START_REQ, ATI_START_CNF, ATI_STOP_REQ, ATI_STOP_CNF,
ATI_EST_REQ(SciUartType), ATI_EST_CNF(DevcoDeleteType),
ATI_REL_REQ, ATI_REL_CNF, ATI_PSTAT_IND(DevcoPstateType),
ATI_PEST_REQ(SciUartType), ATI_PEST_CNF(DevcoResultType,SciUartType),
ATI_PREL_REQ, ATI_PREL_CNF;

" SCI Interactions */
"------------------ */
SciBOpen(SciUartType,SciPathType,InLenType,OutLenType),
SciBClose(SciUartType,SciPathType),
SciDataInd(SciMailReasonType),
SciIRead(SciUartType,SciPathType,Charstring,Int),
SciIWrite(SciUartType,SciPathType,Charstring,Int),
SciITxBufStat(SciUartType),
SciIRxBufStat(SciUartType);

" SMSPP Service Interactions */
"------------------------- */
servSmsReadData(categoryType,readModeType,pageNumber),
servSmsWriteData(locModeType,pageNumber),
servSmsDelete(delModeType,pageNumber),
servSmsSendData(msgRefStar,CharStar),
servSmsSetRouting(msgIndParamTypeStar),
servSmsGetSetting(Octet,CharStar),
servSmsPutSetting(Octet,CharStar),
Procedure `setTextModeParams` `settextmodeparam.spd`

Procedure `setToFactoryConfig` `settofactorydefinedconfig.spd`

Procedure `signalQuality` `signalquality.spd`

Procedure `writeCharSet` `writecharset.spd`

Procedure `writeCmdLineEditChar` `writecmdlineeditchar.spd`

Procedure `writeCmdLineTermChar` `writecmdlinetermchar.spd`

Procedure `writeDceRespFormat` `writedceresponseformat.spd`

Procedure `writeEchoMode` `writeechomode.spd`

Procedure `writeMsgRcvIndProc` `writemsgrcvind.spd`

Procedure `writeMsgToMemory` `writemsgtomem.spd`

Procedure `writePhbookEntry` `writephbookentry.spd`

Procedure `writeResCodeHeadPrsnParams` `writeresultcodeheadparam.spd`

Procedure `writeRespFormatChar` `writeresponseformatchar.spd`

Procedure `writeResultCodeSuppMode` `writeresultcodesuppmode.spd`

Process `PARSER` `parser.spr`

Procedure `decide_msg_text_len` `decidemsgtxtlen.spd`
Procedure ati_emptyOutQue ati_emptyoutque.spd
Procedure ati_findExistingParity ati_findexistingparity.spd
Procedure ati_handleEcho ati_handleecho.spd
Procedure ati_initForNewBMode ati_initfornewbmode.spd
Procedure ati_readFromOutQue ati_readfromoutque1.spd
Procedure ati_resetByteIO ati_resetbyteio.spd
Procedure ati_resetTimer ati_resettimer.spd
Procedure ati_responseInd ati_responseind.spd
Procedure ati_setTimer ati_settimer.spd
Procedure ati_task_init ati_task_init.spd
Procedure connComplTimeout conncompltimeout.spd
Procedure returnToCommandState returntocommandstate1.spd
Procedure ringTimeout ringtimeout1.spd

Block KERNEL kernel.sbk

KERNELTypesAndSignals

Process EXECUTER executer.spr
### APPENDIX B

### ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATI</td>
<td>AT Interpreter</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time Division Multiple Access</td>
</tr>
<tr>
<td>FDMA</td>
<td>Frequency Division Multiple Access</td>
</tr>
<tr>
<td>BS</td>
<td>Base Station</td>
</tr>
<tr>
<td>BSS</td>
<td>Base Station System</td>
</tr>
<tr>
<td>BTS</td>
<td>Base Transceiver System</td>
</tr>
<tr>
<td>BSC</td>
<td>Base Station Controller</td>
</tr>
<tr>
<td>MSC</td>
<td>Mobile services Switching</td>
</tr>
<tr>
<td>TE</td>
<td>Terminal Equipment</td>
</tr>
<tr>
<td>TA</td>
<td>Terminal Adaptor</td>
</tr>
<tr>
<td>ME</td>
<td>Mobile Equipment</td>
</tr>
<tr>
<td>CLIR</td>
<td>Calling Line Identification Presentation</td>
</tr>
<tr>
<td>RLP</td>
<td>Radio Layer Protocol</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identity Module</td>
</tr>
<tr>
<td>SLAM</td>
<td>Service Library Access Manager</td>
</tr>
<tr>
<td>QMAN</td>
<td>A Queue Manager</td>
</tr>
<tr>
<td>MMI</td>
<td>Man Machine Interface</td>
</tr>
<tr>
<td>CAF</td>
<td>Common Application Framework</td>
</tr>
<tr>
<td>COMA</td>
<td>Configuration Manager</td>
</tr>
<tr>
<td>DEVCO</td>
<td>Device Controller</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
</tbody>
</table>
ABOUT THE AUTHOR

Bülent Önen, who was born in Istanbul in 1971, was graduated from Istanbul Sehremini High School in 1988. In 1992 he finished Computer Engineering department of Istanbul Technical University. Now he is working at Software Design department of Netas. He worked on DRX Rural Exchange project, DECT-FRA project as software engineer.