Technical Document - Confidential

GSM PROTOCOL STACK

G23

SMI – SLIM MAN MACHINE INTERFACE

DETAILED SPECIFICATION

Document Number:	8415.011.99.005
Version:	0.5
Status:	Draft
Approval Authority:	
Creation Date:	1999-Jan-27
Last changed:	2015-Mar-08 by Frank Kaiser
File Name:	8415_011.doc

Important Notice

Texas Instruments Incorporated and/or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and/or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and/or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and/or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and/or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
1999-Jan-27	SAB et al		0.1		1
1999-Feb-08	MS et al		0.2		2
2000-Feb-07	OSE et al		0.3		3
2003-May-08	XGUTTEFE		0.4	Draft	
2004-Nov-08	FK		0,5		

Notes:

1. Initial version
2. English check
3. New template

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000] International Organization for Standardization. Quality management sys-

tems - Fundamentals and vocabulary. December 2000

1.1 References

[C_8410.001] 8410.001.98.102, September 18, 1998
G23 Product Description; Condat

[C_8410.008] 8410.008.98.002, June 15, 1998
GTI Interface Description; Condat

[C_8410.003] 8410.003.98.103, September 09, 1998
Test Facilities Description; Condat

[C_8411.802] 8411.802.99.104; January 29, 1999
ACI – Application Control Interface, Functional Interface Description Part 2: F&D; Condat

[GSM_02.30] ETS March 1995 (Version 4.13.0)
European digital cellular telecommunications system (Phase 2), Man-Machine Interface (MMI) of the Mobile Station (MS); ETSI

[GSM_07.07] ETS 300 916: February 1998 (GSM 07.07 version 5.5.0)
AT command set for GSM Mobile Equipment (ME) ; ETSI

[T_25ter] ITU-T V.25 ter; July 1997
Telecommunication Standardization Sector of ITU, Serial asynchronous automatic dialing and control; ITU

1.2 Abbreviations

ACI	Application Control Interface
FH	Frame Handler
KSD	Key (Stroke) Sequence Decoder
LED	Light Emitting Diode
LCD	Liquid Crystal Display
MMI	Man Machine Interface
NH	Network Handler
PCO	Point of Control and Observation
PEI	Protocol Stack Entity Interface
SI	Supplementary Information
SMI	Slim Man Machine Interface
SSC	Supplementary Service Control String
TAP	Test Application
UDUB	User Determined User Busy
UH	User Handler
USSD	Unstructured Supplementary Service Data

1.3 Terms

---

2 Introduction

G23 is a software package implementing Layers 2 and 3 of the ETSI-defined GSM air interface signaling protocol, and as such represents the part of a GSM mobile station's protocol software which is both, platform and manufacturer independent. Therefore, G23 can be viewed as a building block providing standardized functionality through generic interfaces for easy integration.

The G23 suite of products consists of the following items:

- Layers 2 and 3 for speech & short message services,
- Layers 2 and 3 for fax & data services,
- Application Control Interface,
- Slim MMI [GSM_02.30] and
- Test and integration support tools.

For a detailed reference of G23 components, please refer to the Product Description [C_8410.001]. For detailed information regarding integration into the target system, please refer to the Generic Target Interface [C_8410.008]. For detailed information about the compiling and linking procedure, please refer to the User Guide on the delivery CD.

This Functional Interface Specification describes the behavior of SMI as seen from outside. It gives a general idea of the component. The interfaces of all functions SMI offers and of all functions SMI uses are listed.

The SMI solution does not provide a user interface acceptable for mobile customers, rather it shows the principles of communication with the ACI and controls the mobile for type approval purposes. This SMI is implemented in accordance with GSM 02.30 [GSM_02.30]. The attribute *slim* is used to indicate that only a small subset of the features usually offered by mobiles currently on the market are implemented. A summary of all supported functionalities is found in the annex, see Chapter 7.1.

In general, the man machine interface is necessary to enable a user to operate a mobile. For this purpose, hardware components such as display, keyboard, LED, buzzer, loudspeaker, microphone, etc. are under the control of the SMI to enable easy inputs and outputs. The G23 via the ACI is used on the other hand to gain access to the network.

3 Structure

3.1 Modules

3.1.1 Module emi_ksd

This module implements a decoder for supplementary service control strings which can be used within an MMI to map the procedures defined in GSM 02.30 [GSM_02.30] on the functional interface of the ACI [C_8411.802]. The main task is to recognize the called service and prepare the supplementary information for use in function calls to the functional interface of ACI [C_8411.802].

3.1.1.1 Supplementary Service Control Strings

The GSM 02.30 [GSM_02.30] defines a series of predefined MMI procedures which leads to activation, deactivation, interrogation, registration or erasure of services specified by the service code and described in detail by the supplementary information (SI). A string representing such a procedure is called a supplementary service control string (SSC). In addition to those procedures described in GSM 02.30 [GSM_02.30], there are many manufacturer specific procedures which will be supported by the decoder described in this document but not dealt with supplementary service control at all. Therefore, the entire subset of these strings is called keystroke sequences or key sequences.

A summary of all supported key sequences is found in the annex, see 7.2.

3.1.1.2 Decoding Principles

The module KSD is based on at least three tables which contain information about the key sequences supported. The first table contains all key sequences which are applicable while at least one call is active or held, the second one all sequences which contain passwords and the third one all the rest.

The sequences are identified at the first few characters until the sequences end in total which is indicated by the character '#' or when optional supplementary information is followed. A table entry which may contain supplementary information is marked by a '?' at its end.

Once a sequences is recognized a unique decode function is called for each table entry and the rest of the string is parsed if it matches a predefined pattern. The return value of the decode function indicates whether or not decoding was successful. The decoded supplementary information is passed to the calling routine by appropriate structures. The sequences group identifies the structure used which depends on the incoming key sequences. See 7.2 for a mapping of sequence groups and corresponding structures. A union is used to pass all different structures by at least one type of pointer.

3.1.1.3 Interfacing

The interface of the key sequence decoder offers two independent functions. One is used for decoding the strings and transferring the supplementary information to data types used by the application control interface. The other function is used to hide passwords which are located in the control strings as supplementary information. This chapter provides some useful hints for using this functions.

GLOBAL BOOL ksd_decode (CHAR * inSeq, BOOL inCall, T_KSD_SEQGRP * outSeqGrp, CHAR ** outRestSeq,
T_KSD_SEQPARAM* outSeqParam);

<inSeq>: This is the key sequence which will be decoded, e.g. "*#31#"
<inCall>: Indicates whether the mobile equipment is within call state. This indicator will be used for some specific sequences according to the table in chapter 7.2.
<outSeqGrp>: The sequence group to which the incoming key sequence belongs.
<outRestSeq>: Points to the character after the last decoded character.
<outSeqParam>: Points to a structure which contains the mapped supplementary information, if there is any.

The function returns TRUE if decoding was successful, otherwise FALSE.

Supplementary information may consist of strings as well, e.g. passwords -> "**03*353*PW*PW*PW#" -> "**03*353*1234*5678*9012#". To simplify the memory management of the decoder it modifies the incoming strings for separating such substrings by replacing the separators '*' or '#' by '\0'. This could be happen even when decoding fails but further parameters are already decoded. Therefore, to prevent the decoder from destroying the original string it is recommended to pass a copy of it.

GLOBAL void ksd_getPwdHidden (CHAR * inSeq, CHAR replace);

<inSeq>: This is the key sequence which will be manipulated.
<replace>: This is the character which is used for replacing passwords.

This function is used to hide passwords entered during input of a keystroke sequences. The user can select the character which is used instead of the original password characters, e.g. '*' or '.'. The keystroke sequence which is passed as parameter will be changed by this function so the user has to pass a copy of the original string if it will be used otherwise.

String before function call	String after function call:	
**03*353*1234*56	**03*353*....*..	(passwords present)
*67*123456	*67*123456	(no password present)

This function can be used to perform a simple input for the user and must be called every time input changes and the user is enabled to enter keystroke sequences containing passwords.

3.1.2 Module emi_utl

This module defines utility functions for the *emi_ksd* module of the SMI. This includes functions for processing of phone number strings and functions for converting strings into numerical values.

3.1.3 Module mmi_men

This module implements functions for menu processing in display areas supported by the module *mmi_wm*. Up to five menus can be handled in parallel. There are functions for creating and destroying menus, performing menu selections and navigation. This module only makes single-column menus available. Cascading of menus must be realized by the client by using two or more of the menus offered by *mmi_men*.

This module is responsible for the visual representation of menus on the LCD. Menu labels are always one-line and the selected menu item is marked by a '>' in front:

```
    MENU ITEM 1
>   MENU ITEM 2
    MENU ITEM 3
```

Note: Only a small subset or all functionalities of SMI is accessible via menu, mostly the input of supplementary service control strings (SSC) is needed to achieve a wanted effect.

3.1.4 Module mmi_sld

This module defines functions for entering values over a slider bar controlled via suitable keys on the keypad, e.g. volume control. Sliders are connected to display areas which are supported by the module *mmi_wm*. Up to three sliders could be handled in parallel. There are functions for creating and destroying sliders, performing value selection and decreasing or increasing of slider value.

This module is responsible for the visual representation of slider bars on the LCD:

```
---+----------  (low value)
-------+-------  (mid value)
----------+--  (high value)
```

3.1.5 Module mmi_wm

This module defines functions for the window management on the LCD. Up to thirteen windows can be handled parallely. There are functions for creating, destroying, showing and hiding of windows. The stacking order of all created windows could also be manipulated. Of course there are functions to clear the content of a window and to write a new one to it. The window manager is only able to support character orientated displays with cursor functionality.

Requests for changing the display content will be performed by sending the MMI_DISPLAY_REQ primitive to PL. Because updating the LCD content is very time consuming a virtual LCD is supported by the window manager. When the LCD has to be refreshed only changes will be carried out, not clearing the whole content and writing it again.

3.1.6 Module smi_aca

This functional block is the central manager for all network related aspects. ACA has knowledge of the most important internal states concerning the NH and is therefore responsible for reactions on network related inputs. It uses the functional interface of the command handler within the ACI to gain access to network related functionalities and the underlying hardware. It is also notified of appropriate events from these entities. Events which cannot be handled by this functional block are preprocessed and forwarded to the UH.

3.1.7 Module smi_cal

This module handles all call related aspects of the SMI. In detail, the duties are setting up, accepting and releasing calls. Incoming calls are signaled by a call to a suitable call back function defined in the functional interface specification of ACI [C_8411.802] and implemented by this module.

Aside from the previously mentioned basic functionalities, *smi_cal* manages an abbreviated dialing phonebook as long as this feature is not yet implemented by ACI. Up to 10 phone numbers can be stored and restored by passing the index of the memory location. The abbreviated dialing phonebook is deleted when the entity is reset.

This module is able to handle call related supplementary services such as call waiting, call hold and multiparty services in accordance with GSM 02.30 [GSM_02.30] as well as to pass unstructured supplementary service data transparently to the network.

3.1.8 Module smi_csf

This module defines the customer specific functionalities for SMI. Most of them are explained in detail in Chapter 4.3.

For the PC version of the GSM protocol stack, there is a simulation of the LCD driver normally located in the libraries of the lower layer supported by Texas Instruments France. This simulation is used to display a copy of the current LCD content in the corresponding windows of TAP and/or PCO.

3.1.9 Module smi_dmy

This module implements all callback functions not yet analyzed by the SMI.

3.1.10 Module smi_inp

This module defines functions for the processing of entered characters from the keypad and supports a simple line editor. Editors are connected to display areas supported by the module *mmi_wm*. Up to three editors can be handled in parallel fashion. There are functions for creating and destroying editors, for setting a predefined input string, adding a single character to the input line and getting back the whole edited string.

The editor offers the user only one single edit line, deleting of a single character using a backspace mechanism and automatic side scrolling. The input mode of the editor can be set according to any useful combination of the following flags:

IT_NUMERIC	input of phone numbers is enabled, the following characters are permitted: 0 1 2 3 4 5 6 7 8 9 * # + - <SPACE>
IT_ALPHA	input of any character is enabled
IT_ECHO	all entered characters will be echoed on the display
IT_HIDDEN	all characters entered will be represented by a '*'

3.1.11 Module smi_lng

This module defines functions for language support. Any output string is identified by the current language and a unique text string identifier.

Note: Once a string is displayed on the screen, no update is performed when language will be changed by the user. The output in a window will be refreshed with respect to the language; at the moment a new text string is written to this window.

3.1.12 Module smi_pei

This module defines the protocol stack entity interface for the SMI. For a complete list of all functions called by the ACI when it is triggered by the frame, see Chapter 4.3.

3.1.13 Module smi_reg

This module defines the registration/deregistration handling for the SMI component of the mobile station. The following three modes are supported: manual, automatic and manual followed by automatic registration. Aside from this, the power on/off routines for the GSM protocol stack are implemented in this module. Every time the mobile is powered on, automatic registration is started without any influence from the user. For manual registration, there will be functions for requesting a list of all operators in scope.

3.1.14 Module smi_sec

This module defines the security and SIM handling for the component SMI of the mobile station. It supports the PIN and PUK handling as well as the registration of new facility passwords.

3.1.15 Module smi_sms

This module is responsible for the management of short messages and cell broadcast messages. The following procedures are supported: submit, submit from memory, write to memory, read from memory, delete from memory and list all messages in memory.

For an SMS, submit the following settings are assumed and can not be dynamically altered:

protocol identifier: 0

data coding scheme: 0

status report always requested

validity period relative/not present

The module *smi_sms* has its own communication line to the user handler represented by the module *smi_ui* while all other information is passed via the communication line held by the *smi_aca* module. This separation was made to simplify the structure of SMI and to ensure an easy understanding.

Note: Cell broadcast messages will not be stored in memory due to the restrictions of the GSM protocol stack. The same reason that no user data header is supported for submitted messages. User data header for received messages is supported by the GSM protocol stack but not by SMI.

3.1.16 Module smi_ss

This module defines the supplementary service functions for the SMI. The following list provides an overview while details can be gathered from the summary in the annex, see 7.2.:

CLIP, CLIR, COLP
Call Forwarding
Facility Lock
Call Waiting

3.1.17 Module smi_tim

This module defines the timer handling functions for the Slim Man Machine Interface. Every time a timeout occurs this event will be entered in a FIFO and the routine called will return immediately from interrupt context without a final handling of this timeout. The final handling will happen when the next primitive arrives. The first action is to look up the timer FIFO and call the corresponding timeout function if necessary. To ensure that a primitive is sent to the entity, the entity sends itself an empty primitive when called in the interrupt context.

The module *smi_tim* implements functions for starting, dynamic configuration and timeout handling of timers.

3.1.18 Module smi_ui

This module is the central manager for the user interface. It has knowledge of the most important internal states concerning the UH and is therefore responsible for reactions on user, hardware or network related inputs which will be preprocessed by ACA or SMS. It communicates with the functional block ACA and SMS to gain access to network related functionalities and the underlying hardware as well as it is notified of appropriate events from these entities. This module combines all resources supported by the modules of the UH to a structure that implements the man machine interface the user is able to operate.

3.2 Headers

The modules include several header files. Header files which are changeable by the user are marked (*). These header files are used to integrate the protocol stack entities into a specific target system.

Header aca (*)

This header contains several definitions and declarations for the network handler of SMI.

Header aci_cmh

This header contains all definitions for the functional interface of the ACI [GSM_02.30].

Header custom (*)

This header defines global constants for the integration of the protocol stack entity into a specific target system. The user may define the identifier of the communication resource, the supported traces, the communication method (by copying primitives or by exchanging references of primitives) , the custom specific primitive header, etc.

Header cus_asb

This header defines constants shared by ACI and SMI. Once this header has to be changed for any reason, the ACI component has to be recompiled as well.

Header cus_smi (*)

This header defines customer specific constants related to timer control, language support and keypad mapping.

Header cnf_smi (*)

This header defines constants for the dynamic configuration of SMI.

Header emi_ksd (*)

This header contains definitions and declarations for the decoding of key sequences.

Header emi_utl (*)

This header defines utility functions for the SMI component of the GSM protocol stack.

Header gsm

This header contains global definitions for all protocol stack entities. Depending on the definitions in custom.h, many options and traces are defined in this header.

Header lcd

This header contains all function prototypes for the LCD driver.

Header pei (*)

Prototypes for the protocol stack entity interface are defined in this header. Some parameters and the return type of this function are changeable by the user.

Header prim

Constants for primitives are defined and service access point dependent primitive header files are included (p_mph.h., p_dl.h and p_rr.h).

Header smi (*)

This header contains several definitions and declarations for the user handler and the frame handler of SMI.

Header stdarg

This header defines ANSI-style macros for accessing arguments of functions which take a variable number of arguments. It is provided by the target compiler.

Header stddefs

The header contains several standard definitions used by the protocol stack entities.

Header stdlib

This header is a the standard library header from the target compiler.

Header string

This header is a the standard string header from the target compiler. It defines string and memory functions.

Header tok

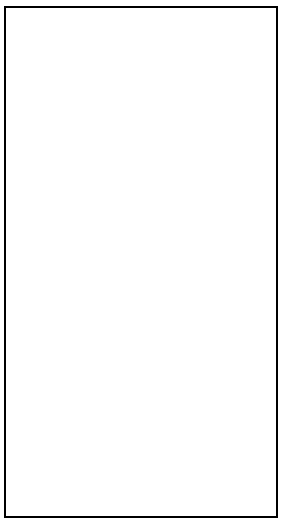
The prototypes and some constants for the parse function of the TOK module are defined in this header.

Header vsi (*)

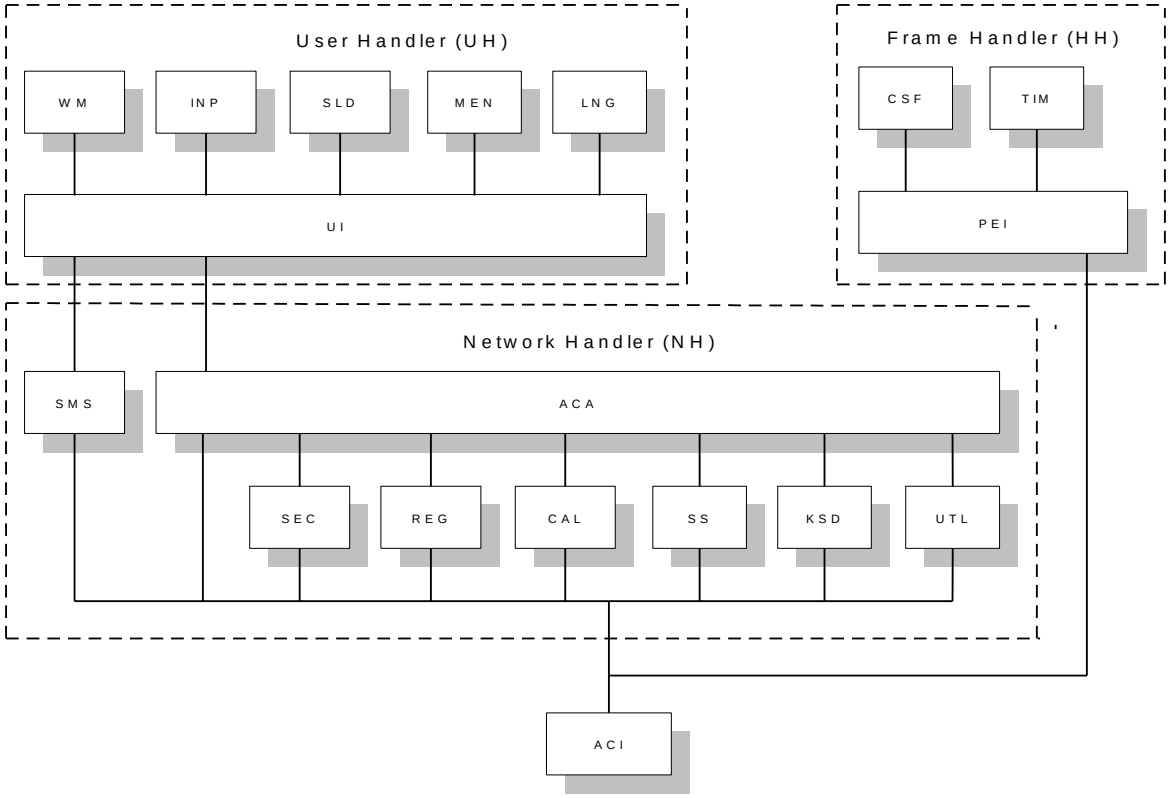
Prototypes for the virtual system interface are defined in this header. Some parameters and the return types of these functions are changeable by the user for integration into a specific target system.

3.3 Functional Structure

The SMI uses the functional interface of the ACI [C_8411.802] to control the network related functions of the GSM protocol stack and the underlying hardware. Via this interface, SMI is also notified about all events from these entities which cannot be processed by the lower layer. SMI and ACI are both linked to one entity. The resulting entity is able to run on the target system and on a PC.



SMI is divided into the three blocks user handler (UH), network handler (NH) and frame handler (FH). UH is the manager for all user related functionalities while NH takes care of all network related aspects as well as maintaining communication with the ACI and by means of this gains access to the GSM protocol stack and the underlying hardware. All relevant function calls to the protocol stack entity interface of ACI are routed transparently to FH which is mainly responsible for (de)initialization, dynamic configuration and timer control. UH, NH and HH consist of several subblocks which are responsible for different functionalities. See Chapter 3.1 for a brief description. The following picture shows the architecture of the SMI and the functional blocks defined.



3.4 Data Structure

The central data structure which is used to show the internal state of the entire SMI subentity is called T_MMI_DATA and there will be only one instance of this in the whole system. All modules have full access to this structure for an easy implementation even when the principle of hiding information is broken.

```
typedef struct
{
        UBYTE       state;
        UBYTE       serviceType;
        BOOL        extDisplay;
}
T_MMI_DATA;
```

The state of the SMI has one of the following values:

```
MMI_OFF                 the SMI is switched off
MMI_ON                      the SMI is switched on
MMI_WAIT_FOR_PIN    the PIN verification process is in progress
MMI_WAIT_FOR_PUK    the PUK verification process is in progress
MMI_IDLE                the SMI is in idle mode
MMI_WAIT_FOR_REG the SMI waits for end of registration process
MMI_WAIT_FOR_DEREG      the SMI waits for end of deregistration
process
```

The following services are supported:

```
SRV_NONE     out of service
SRV_LIMITED  limited service (registered but no  SIM or SIM but no authenti-
cation)
```

SRV_NORMAL normal service: SIM inserted and authorized

3.5 Interface

The control of all network related functionalities is performed via the functional interface of the ACI. A detailed specification of this interface is provided in a separate document [C_8411.802].

Aside from the network related functionalities, the interface of the ACI [C_8411.802] also supports some hardware related aspects. Keypad indications are signaled to the SMI, whereas audio path aspects and backlight are both requested by SMI. Because this concept of hardware access is only a temporary solution, there are already two primitives sent by SMI: MMI_DISPLAY_REQ and MMI_AUDIO_TONE_REQ. No primitives are received by SMI.

The control of all frame related functionalities is performed via the functional interface of the module smi_pei. Functional calls from the frame to the protocol stack entity interface of the ACI are routed to the protocol stack subentity interface of SMI where necessary.

4 Configuration

4.1 Static Configuration

Static configuration is used to influence the behavior of the protocol stack entity at compile time. Once the object is built, further changes are not possible.

TSLIDER_VALUE

Timeout value for switching of the slider, defined in file cus_smi.h.

TBACKLIGHT_VALUE

Timeout value for switching of the backlight, defined in file cus_smi.h.

T_SHIFTKEY_VALUE

Timeout value for switching the keypad mode from shifted back to normal, defined in file cus_smi.h.

MAX_SMI_TIMER

Maximum number of timers available in the timer pool, defined in file cus_smi.h.

KP_OK

Basic key code delivered by ACI when pressing <OK>, defined in file cus_smi.h.

KP_CLEAR

Basic key code delivered by ACI when pressing <CLEAR>, defined in file cus_smi.h.

KP_ON_HOOK

Basic key code delivered by ACI when pressing <END>, defined in file cus_smi.h.

KP_OFF_HOOK

Basic key code delivered by ACI when pressing <SEND>, defined in file cus_smi.h.

KP_UP

Basic key code delivered by ACI when pressing <UP>, defined in file cus_smi.h.

KP_DOWN
Basic key code delivered by ACI when pressing <DOWN>, defined in file cus_smi.h.

KP_SHIFT
Basic key code delivered by ACI when pressing <SHIFT>, defined in file cus_smi.h.

KP_F1
Basic key code delivered by ACI when pressing <F1>, defined in file cus_smi.h.

KP_F2
Basic key code delivered by ACI when pressing <F2>, defined in file cus_smi.h.

KP_POWER
Basic key code delivered by ACI when pressing <POWER>, defined in file cus_smi.h.

KP_INC
Basic key code delivered by ACI when pressing <INC>, defined in file cus_smi.h.

KP_DEC
Basic key code delivered by ACI when pressing <DEC>, defined in file cus_smi.h.

KP_STAR
Basic key code delivered by ACI when pressing '*', defined in file cus_smi.h.

KP_HASH
Basic key code delivered by ACI when pressing '#', defined in file cus_smi.h.

TBACKLIGHT
Identity for backlight timer, defined in smi.h.

TSLIDER
Identity for slider timer, defined in smi.h.

TSHIFTKEY
Identity for shifted key timer, defined in smi.h.

TMAX
Maximum value of any timer identity, defined in smi.h

SMI_TSTART
Macro for setting of timer start function, defined in smi.h.

EMERGENCY SEQUENCE
Setting of the phone number which will be called in case of an emergency. There is only one static emergency call code supported.

MAX_ABBR_DIALING_NUM

Maximum number of locations in abbreviated dialing phone book, defined in smi_cal.c.

Arrays **statText** and **text**
Definition of all texts visible on the LCD in five different languages, defined in smi_lng.c.

Numerical variable **aktLanguage**
Definition of the currently used language after initializing the SMI.

4.2 Dynamic Configuration

Dynamic configuration means to change the behavior of the protocol stack entity at run-time. This is carried out by sending a string with a dedicated format as described in Test Facilities [C_8410.003]. The dynamic configuration string is a parameter of the pei_config () function which is part of the protocol stack entity interface.

TIMER_SET = <timer, value>
The timer mode TIMER_SET defines a new timer value instead of the original start value.

TIMER_RESET = timer
The default timer mode is TIMER_RESET which does not manipulate the start value.

TIMER_SPEED_UP = <timer, factor>
TIMER_SPEED_UP is used to speed up a timer by the given factor. The start value is divided by the factor. The minimum time is one unit.

TIMER_SLOW_DOWN = <timer, factor>
The opposite mode is TIMER_SLOW_DOWN. The start value is increased by the given factor.

TIMER_SUPPRESS = <timer>
TIMER_SUPPRESS is used to suppress the timer start.

KEY_SEQUENCE = <string>
KEY_SEQUENCE is used to pass supplementary service control strings to the SMI.

4.3 Custom Specific Functions

Custom specific functions are implemented in the module smi_csf.c. It is acceptable to replace the functions in this module with functions from the customer. It is not acceptable to change the parameters of the functions.

The idea behind these custom specific functions is to have a mechanism to configure the protocol stack entity at run-time by a source outside the protocol stack entity, for example a non-erasable memory.

GLOBAL BOOL csf_init_timer (void)

The function initializes the timer pool. The timer pool allocates a number of timer resources. This timer resources are allocated to instances on demand.

GLOBAL void csf_close_timer (void)

All timer resources are closed. This function is carried out during shutdown or reset.

GLOBAL void _csf_alloc_timer (UBYTE id, T_VSI_TVALUE value, T_VSI_THANDLE * handle)
The function allocates one timer of the timer pool and starts this timer.

GLOBAL void csf_free_timer (T_VSI_THANDLE handle)

The function frees one timer. The timer is stopped and returned to the timer pool.

GLOBAL BOOL _csf_vdb_timeout (T_VSI_THANDLE handle, USHORT * timer)
After time-out the corresponding instance is searched. The timer is returned to the timer pool.

GLOBAL T_PEI_RETURN _pei_init (void)

This function initializes the SMI. It is called whenever ACI is initialized itself under the control of the frame.

GLOBAL T_PEI_RETURN _pei_primitive (T_PRIM *prim)

This function processes protocol stack entity specific primitives. It is called whenever ACI itself was informed about a new incoming primitive by the frame and cannot process this primitive on its own.

GLOBAL T_PEI_RETURN _pei_timeout (T_VSI_THANDLE handle)

This function is used to signal a timeout condition of the timer identified by its handle.

GLOBAL T_PEI_RETURN _pei_exit (void)

This function closes all resources and terminates the SMI process. It is called whenever ACI is terminated itself under the control of the frame.

GLOBAL T_PEI_RETURN _pei_config (T_PEI_CONFIG inString, T_PEI_CONFIG outString)

This function configures the SMI at run-time.

GLOBAL T_PEI_RETURN _pei_monitor (void ** monitor)

This function monitors physical parameters of SMI.

GLOBAL UBYTE csf_getVolume (T_volType volType)

This function reads the volume value of the given volType out of the EEPROM area of the mobile.

GLOBAL void csf_setVolume (T_volType volType, UBYTE volume)

This function writes the volume value of the given volType into the EEPROM area of the mobile.

GLOBAL USHORT csf_getLanguage (void)

This function gets the language identity from the EEPROM area of the mobile.

GLOBAL void csf_setLanguage (USHORT lng)

This function sets the language identity in the EEPROM area of the mobile.

GLOBAL void csf_getPanelInfo (USHORT * LCD_DX, USHORT * LCD_DY,

USHORT * KeyboardType, BOOL * extDisplay)

This function requests the screen size of the LCD and the type of keyboard used.

4.4 Build a Entity

4.4.1 Compiling SMI

For control of compilation processes, there are the following preprocessor definitions which must be set according to the customer's needs: SIMULATION, OPTION_MULTITHREAD, SHARED_VSI, SIMULATE_LCD.

The following preprocessor definition always has to be set independently of the customer's needs: SMI.

The struct member alignment must be set to the value of 1 byte.

4.4.2 Linking SMI

SMI is not an independent entity which can be built alone. SMI and ACI are always linked together to one entity named "SMI". To ensure compatibility of both libraries care has to be taken in the compilation process. Both libraries must be generated using concurrent settings.

5 Monitoring

The monitor struct includes relevant physical parameters of the protocol stack entity. The parameters are updated continuously. This way the environment always has the possibility of accessing parameters of the protocol stack. These parameters are used to create monitor reports to a display or a test system, to create statistical data, etc. outside the functionality of a protocol stack but with access to protocol stack parameters. It is acceptable to read the parameters of the monitor struct but it is absolutely not acceptable to write to the monitor struct. The first parameter of the monitor struct is the version of the protocol stack entity.

The following monitor struct is defined for the protocol stack entity SMI:

```
typedef struct
{
        T_VERSION    *version;
} T_MONITOR;
```

6 Resources

6.1 Timer

SMI defines three timers. They are responsible for switching off backlight and slider and for returning to normal keypad mode after <SHIFT> was pressed. The timer values can be defined in the header file cus_smi.h

6.2 Memory

SMI needs no permanent dynamic allocated memory. All data is stored in memory areas allocated during compile time. For sending primitives, memory out of the primitive memory pool is temporarily allocated.

7 Annex

7.1 SMI Feature List

Service	Aspects	Comments
Security	PIN and PIN2 input PIN and PIN2 change PIN and PIN2 unblocking Status information return codes Presentation of IMEI	High priority, completed Via SSC, high priority, completed High priority, completed Via SSC, mid priority, postponed Via SSC, mid priority, postponed
Call Control	Mobile originated voice calls Mobile terminated voice calls Emergency calls Saving of emergency call codes Abbreviated Dialing DTMF Call progress indication	High priority, completed High priority, completed High priority, completed Via SSC, low priority, postponed Via SSC, low priority, completed Mid priority, completed High priority, completed
Network Registration	Automatic registration Manual registration Registration progress indication	High priority, completed High priority, completed High priority, completed
Supplementary Services	Handling of SS while call established Handling of SS via SSC	High priority, completed High priority, completed
SMS	Sending SMs Receiving SMs Reading SMs Writing SMs Receiving CBMs Reading CBMs Full memory indication	High priority, completed High priority, completed High priority, completed High priority, completed High priority, completed High priority, completed High priority, completed
Fax / Data	---	Low priority, postponed
General	Field strength indication Battery charge indication Volume control Interpretation of user defined SSC Call charge units meter Language support Switch between keypad/AT command control Numerical, alphanumerical editor Hide passwords during input of SSC	Mid priority, completed Mid priority, completed Mid priority, completed High priority, completed Mid priority, postponed Low priority, completed High priority, completed High priority, completed Low priority, completed

7.2 Key Sequences Supported

The following table provides an overview of the keystroke sequences supported, the corresponding sequence group and the decoded parameter. In a call to a function of the ACI, the decoded parameter will be passed using a pointer to a union type. The sequence group is used to identify the correct data type this pointer referenced.

Keystroke sequence	Description	Sequence group	Decoded parameter type
Combination of 0 1 2 3 4 5 6 7 8 9 A B C D * # + -	Phone number, used for dialing	SEQGRP_DIAL	T_KSD_DIAL
Unknown supplementary service control procedure	Unstructured supplementary service data (USSD)	SEQGRP_USSD	---
0	Releases all held calls or set user determined user busy (UDUB) for a waiting call (only applicable when within call state)	SEQGRP_SND_CHLD	T_KSD_CHLD
1	Releases all active calls (if any exist) and accepts the other (held or waiting) call (only applicable when within call state)	SEQGRP_SND_CHLD	T_KSD_CHLD
1X	Releases a specific active call X (only applicable when within call state)	SEQGRP_SND_CHLD	T_KSD_CHLD
2	Places all active calls (if any exist) on hold and accepts the other (held or waiting) call (only applicable when within call state)	SEQGRP_SND_CHLD	T_KSD_CHLD
2X	Places all active calls on hold except call X with which communication shall be supported (only applicable when within call state)	SEQGRP_SND_CHLD	T_KSD_CHLD
3	Adds a held call to the conversation (only applicable when within call state)	SEQGRP_SND_CHLD	T_KSD_CHLD
*77742*PW#	Locks SIM card	SEQGRP_ACT_SIM_LOCK	T_KSD_CLCK
"#77742*PW#	Unlock SIM card	SEQGRP_DEACT_SIM_LOCK	T_KSD_CLCK
*333*PW*BS#	Activates outgoing barring services	SEQGRP_ACT_OUT_BARR_SERV	T_KSD_CLCK
#333*PW*BS#	Deactivates outgoing barring services	SEQGRP_DEACT_OUT_BARR_SERV	T_KSD_CLCK
*#333#	Interrogates outgoing barring services	SEQGRP_INTRGT_OUT_BARR_SERV	T_KSD_CLCK_QUERY
*332*PW*BS#	Activates BAOIC except home	SEQGRP_ACT_BAOIC_EXC_HOME	T_KSD_CLCK
#332*PW*BS#	Deactivates BAOIC except home	SEQGRP_DEACT_BAOIC_EXC_HOME	T_KSD_CLCK
*#332#	Interrogates BAOIC except home	SEQGRP_INTRGT_BAOIC_EXC_HOME	T_KSD_CLCK_QUERY
*331*PW*BS#	Activates BAOIC	SEQGRP_ACT_BAOIC	T_KSD_CLCK
#331*PW*BS#	Deactivates BAOIC	SEQGRP_DEACT_BAOIC	T_KSD_CLCK
*#331#	Interrogates BAOIC	SEQGRP_INTRGT_BAOIC	T_KSD_CLCK_QUERY
*330*PW*BS#	Activates all barring services	SEQGRP_ACT_ALL_BARR_SERV	T_KSD_CLCK
#330*PW*BS#	Deactivates all barring services	SEQGRP_DEACT_ALL_BARR_SERV	T_KSD_CLCK
*#330#	Interrogates all barring services	SEQGRP_INTRGT_ALL_BARR_SERV	T_KSD_CLCK_QUERY
*33*PW*BS#	Activates BAOC	SEQGRP_ACT_BAOC	T_KSD_CLCK
#33*PW*BS#	Deactivates BAOC	SEQGRP_DEACT_BAOC	T_KSD_CLCK
*#33#	Interrogates BAOC	SEQGRP_INTRGT_BAOC	T_KSD_CLCK_QUERY
*353*PW*BS#	Activates incoming barring services	SEQGRP_ACT_IN_BARR_SERV	T_KSD_CLCK
#353*PW*BS#	Deactivates incoming barring services	SEQGRP_DEACT_IN_BARR_SERV	T_KSD_CLCK
*#353#	Interrogates incoming barring services	SEQGRP_INTRGT_IN_BARR_SERV	T_KSD_CLCK_QUERY

*351*PW*BS#	Activates BAIC roaming	SEQGRP_ACT_BAIC_ROAM	T_KSD_CLCK
#351*PW*BS#	Deactivates BAIC roaming	SEQGRP_DEACT_BAIC_ROAM	T_KSD_CLCK
*#351#	Interrogates BAIC roaming	SEQGRP_INTRGT_BAIC_ROAM	T_KSD_CLCK_QUERY
*35*PW*BS#	Activates BAIC	SEQGRP_ACT_BAIC	T_KSD_CLCK
#35*PW*BS#	Deactivates BAIC	SEQGRP_DEACT_BAIC	T_KSD_CLCK
*#35#	Interrogates BAIC	SEQGRP_INTRGT_BAIC	T_KSD_CLCK_QUERY
*03*333*PW*PW*PW#	Registers password outgoing barring	SEQGRP_REGPWD	T_KSD_CPWD
*03*332*PW*PW*PW#	Registers password BAOIC except home	SEQGRP_REGPWD	T_KSD_CPWD
*03*331*PW*PW*PW#	Registers password BAOIC	SEQGRP_REGPWD	T_KSD_CPWD
*03*330*PW*PW*PW#	Registers password all barring	SEQGRP_REGPWD	T_KSD_CPWD
*03*33*PW*PW*PW#	Registers password BAOC	SEQGRP_REGPWD	T_KSD_CPWD
*03*353*PW*PW*PW#	Registers password incoming barring	SEQGRP_REGPWD	T_KSD_CPWD
*03*351*PW*PW*PW#	Registers password BAIC roaming	SEQGRP_REGPWD	T_KSD_CPWD
*03*35*PW*PW*PW#	Registers password BAIC	SEQGRP_REGPWD	T_KSD_CPWD
*03**PW*PW*PW#	Registers password for all services	SEQGRP_REGPWD	T_KSD_CPWD
**03*333*PW*PW*PW#	Registers password outgoing barring	SEQGRP_REGPWD	T_KSD_CPWD
**03*332*PW*PW*PW#	Registers password BAOIC except home	SEQGRP_REGPWD	T_KSD_CPWD
**03*331*PW*PW*PW#	Registers password BAOIC	SEQGRP_REGPWD	T_KSD_CPWD
**03*330*PW*PW*PW#	Registers password all barring	SEQGRP_REGPWD	T_KSD_CPWD
**03*33*PW*PW*PW#	Registers password BAOC	SEQGRP_REGPWD	T_KSD_CPWD
**03*353*PW*PW*PW#	Registers password incoming barring	SEQGRP_REGPWD	T_KSD_CPWD
**03*351*PW*PW*PW#	Registers password BAIC roaming	SEQGRP_REGPWD	T_KSD_CPWD
**03*35*PW*PW*PW#	Registers password BAIC	SEQGRP_REGPWD	T_KSD_CPWD
03PW*PW*PW#	Registers password for all services	SEQGRP_REGPWD_ALL_SERV	T_KSD_CPWD
**042*PW*PW*PW#	Changes PIN2	SEQGRP_REGPWD	T_KSD_CPWD
**04*PW*PW*PW#	Changes PIN	SEQGRP_REGPWD	T_KSD_CPWD
**052* t.b.d	Unblocks PIN2	SEQGRP_UNBLCK_PIN2	t.b.d
**05* t.b.d	Unblocks PIN	SEQGRP_UNBLCK_PIN	t.b.d
#*91*0#	MS off	SEQGRP_MS_OFF	---
#*91*1#	MS on	SEQGRP_MS_ON	---
#*43*0#	Hooks off	SEQGRP_HOOK_OFF	---
#*43*1#	Hooks on	SEQGRP_HOOK_ON	---
#*88*DTMF	DTMF	SEQGRP_DTMF	T_KSD_DTMF
#*92*KEYCODE#	Keypad indication	SEQGRP_KEYPAD_IND	T_KSD_KEYPAD_IND
#*93#	Shows call table	SEQGRP_SHOW_CALL_TABLE	---
#*95*IDX*DN#	Sets phone number used for abbreviated dialing	SEQGRP_SET_ABBR_DIAL	T_KSD_ABBR_DIAL
*1#	Restarts registration	SEQGRP_START_REGISTER	---
*1*0#	Sets automatic registration	SEQGRP_CHANGE_REGISTER	---
*1*1#	Sets manual registration	SEQGRP_CHANGE_REGISTER	---
*1*4#	Sets manual followed by automatic registration	SEQGRP_CHANGE_REGISTER	---
MANUAL	Sets manual registration	SEQGRP_SET_REGISTER	T_KSD_COPS
AUTO	Sets automatic registration	SEQGRP_SET_REGISTER	T_KSD_COPS
SPLMN	Starts cell selection for a selected PLMN	SEQGRP_START_REGISTER	T_KSD_COPS

#*94*0#	Sends short message	SEQGRP_SMS_SEND	---
#*94*1#	Sends short message from storage	SEQGRP_SMS_SEND_FROM_MEM	---
#*94*2#	Writes short message to memory	SEQGRP_SMS_WRITE	---
#*94*3#	Deletes short message	SEQGRP_SMS_DELETE	---
#*94*4#	Reads short message	SEQGRP_SMS_READ	---
#*94*5#	Lists short messages	SEQGRP_SMS_LIST	---
*002*DN*BS*T#	Activates all CF	SEQGRP_ACT_ALL_CF	T_KSD_CCFC
#002*DN*BS*T#	Deactivates all CF	SEQGRP_DEACT_ALL_CF	T_KSD_CCFC
**002*DN*BS*T#	Registers all CF	SEQGRP_REG_ALL_CF	T_KSD_CCFC
##002*DN*BS*T#	Erases all CF	SEQGRP_ERASE_ALL_CF	T_KSD_CCFC
*#002#	Interrogates all CF	SEQGRP_INTRGT_ALL_CF	T_KSD_CCFC_QUERY
*004*DN*BS*T#	Activates all conditional CF	SEQGRP_ACT_ALL_COND_CF	T_KSD_CCFC
#004*DN*BS*T#	Deactivates all conditional CF	SEQGRP_DEACT_ALL_COND_CF	T_KSD_CCFC
**004*DN*BS*T#	Registers all conditional CF	SEQGRP_REG_ALL_COND_CF	T_KSD_CCFC
##004*DN*BS*T#	Erases all conditional CF	SEQGRP_ERASE_ALL_COND_CF	T_KSD_CCFC
*#004#	Interrogates all conditional CF	SEQGRP_INTRGT_ALL_COND_CF	T_KSD_CCFC_QUERY
*21*DN*BS*T#	Activates unconditional CF	SEQGRP_ACT_UNCOND_CF	T_KSD_CCFC
#21*DN*BS*T#	Deactivates unconditional CF	SEQGRP_DEACT_UNCOND_CF	T_KSD_CCFC
**21*DN*BS*T#	Registers unconditional CF	SEQGRP_REG_UNCOND_CF	T_KSD_CCFC
##21*DN*BS*T#	Erases unconditional CF	SEQGRP_ERASE_UNCOND_CF	T_KSD_CCFC
*#21#	Interrogates unconditional CF	SEQGRP_INTRGT_UNCOND_CF	T_KSD_CCFC_QUERY
*67*DN*BS*T#	Activates busy CF	SEQGRP_ACT_BUSY_CF	T_KSD_CCFC
#67*DN*BS*T#	Deactivates busy CF	SEQGRP_DEACT_BUSY_CF	T_KSD_CCFC
**67*DN*BS*T#	Registers busy CF	SEQGRP_REG_BUSY_CF	T_KSD_CCFC
##67*DN*BS*T#	Erases busy CF	SEQGRP_ERASE_BUSY_CF	T_KSD_CCFC
*#67#	Interrogates busy CF	SEQGRP_INTRGT_BUSY_CF	T_KSD_CCFC_QUERY
*61*DN*BS*T#	Activates no reply CF	SEQGRP_ACT_NOREP_CF	T_KSD_CCFC
#61*DN*BS*T#	Deactivates no reply CF	SEQGRP_DEACT_NOREP_CF	T_KSD_CCFC
**61*DN*BS*T#	Registers no reply CF	SEQGRP_REG_NOREP_CF	T_KSD_CCFC
##61*DN*BS*T#	Erases no reply CF	SEQGRP_ERASE_NOREP_CF	T_KSD_CCFC
*#61#	Interrogates no reply CF	SEQGRP_INTRGT_NOREP_CF	T_KSD_CCFC_QUERY
*62*DN*BS*T#	Activates not reachable CF	SEQGRP_ACT_NOTREACH_CF	T_KSD_CCFC
#62*DN*BS*T#	Deactivates not reachable CF	SEQGRP_DEACT_NOTREACH_CF	T_KSD_CCFC
**62*DN*BS*T#	Registers not reachable CF	SEQGRP_REG_NOTREACH_CF	T_KSD_CCFC
##62*DN*BS*T#	Erases not reachable CF	SEQGRP_ERASE_NOTREACH_CF	T_KSD_CCFC
*#62#	Interrogates not reachable CF	SEQGRP_INTRGT_NOTREACH_CF	T_KSD_CCFC_QUERY
*#77742#	Interrogates lock status of SIM card	SEQGRP_INTRGT_SIM_LOCK	T_KSD_CLCK_QUERY
*#31#	Interrogates CLIR	SEQGRP_INTRGT_CLIR	---
*31#	Suppresses CLIR	SEQGRP_SUP_CLIR	---
#31#	Invokes CLIR	SEQGRP_INV_CLIR	---
*#30#	Interrogates CLIP	SEQGRP_INTRGT_CLIP	---
*30#	Suppresses CLIP	SEQGRP_SUP_CLIP	---
#30#	Invokes CLIP	SEQGRP_INV_CLIP	---

*#77#	Interrogates COLR	SEQGRP_INTRGT_COLR	---
*77#	Suppresses COLR	SEQGRP_SUP_COLR	---
#77#	Invokes COLR	SEQGRP_INV_COLR	---
*#76#	Interrogates COLP	SEQGRP_INTRGT_COLP	---
*76#	Suppresses COLP	SEQGRP_SUP_COLP	---
#76#	Invokes COLP	SEQGRP_INV_COLP	---
*43*BS#	Activates wait	SEQGRP_ACT_WAIT	---
#43*BS#	Deactivates wait	SEQGRP_DEACT_WAIT	---
*#43#	Interrogates wait	SEQGRP_INTRGT_WAIT	---
*#06#	Presenta IMEI	SEQGRP_PRSNT_IMEI	---
*55#	Activate TTY (bidirectional)	SEQGRP_TTY_SERV	
*55#DN	Activate TTY (bidirectional) for this call	SEQGRP_TTY_SERV	
#55#	Deactivate TTY (bidirectional)	SEQGRP_TTY_SERV	
#55#DN	Deactivate TTY (bidirectional) for this call	SEQGRP_TTY_SERV	
*7767*SPD*NAM*CE#	Sends AT+CBST, GSM 07.07 , chapter 6.7	SEQGRP_SND_CBST	T_KSD_CBST
*7768*IWS*MWS*T1*N2#	Sends AT+CRLP, GSM 07.07 , chapter 6.8	SEQGRP_SND_CRLP	T_KSD_CRLP
*25661*DIR*CMP*DCT*STR#	Sends AT+DS, V25ter [T_25ter], chapter 6.6.1	SEQGRP_SND_DS	T_KSD_DS

Legend:

DN	directory (phone) number
PW	password
BS	bearer service
KEYCODE	any of the following strings: "OK", "CLEAR", "END", "SEND", UP", "DOWN", "SHIFT", "F1", "F2", "POWER", "INC", "DEC", "STAR", "HASH"
IDX	index of abbreviated dialing storage, any value between 0 and 9
DIR:	direction
CMP:	compression negotiation
DCT:	max number of dictionaries
STR:	max string length
IWS:	IWF to MS window size
MWS:	MS to IWF window size
T1:	acknowledgement timer T1
N2:	retransmission attempts N2
SPD:	data rate (speed)
NAM:	bearer service name
CE:	connection element
Note:	DIR, CMP, DCT and STR are numerical values according to ITU-T V25 ter [T_25ter], AT-command +DS DIR IWS, MWS, T1 and N2 are numerical values according to GSM 07.07, AT-command +CRLPSPD SPD, NAM and CE are numerical values according to GSM 07.07, AT-command +CBST NAM

T_KSD_...	structures according to header file emi_ksd.h
SEQGRP_...	enumerator of enumeration T_KSD_SEQGRP defined in emi_ksd.h

Appendices

A. Acronyms

DS-WCDMA Direct Sequence/Spread Wideband Code Division Multiple Access

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000) Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>