Technical Document - Confidential

GSM PROTOCOL STACK

G23

KBD-KEYBOARD DRIVER

FUNCTIONAL SPECIFICATION

Document Number:	8415.009.99.013
Version:	0.9
Status:	Draft
Approval Authority:	
Creation Date:	1998-Sep-10
Last changed:	2015-Mar-08 by XGUTTEFE
File Name:	8415_009.doc

Change History

Date	Changed by	Approved by	Version	Status	Notes
1998-Sep-10	LM et al.		0.1		1
1998-Sep-29	LM et al.		0.2		2
1998-Dec-15	LM et al.		0.3		3
1998-Dec-17	LM et al.		0.4		4
1998-Dec-17	LM et al.		0.5		5
1999-Mar-2	MS et al.		0.6		6
1999-Jun-3	LE et al.		0.7		
2000-Feb-4	UB et al.		0.8		

2003-May-13	XINTEGRA		0.9	Draft	

Notes:

1. Initial version
2. Editorial, typernatic settings handling added
3. Balanced with document 8415.012
4. Editorial, KeyConfig functions template
5. Consistency check/Submitted
6. New Template

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000] International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000

1 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 [02.30] and
- Test and integration support tools.

This document describes the functional interface of the G23 Keyboard driver interface. This driver is used as an interface between applications handling keyboard inputs from keyboard hardware.

The driver takes care of key bouncing and signals key status changes to the process using it if desired (key pressed, key released, key repeat). All keys are represented by a unique key code. This key code is used to identify the key which status has changed. The definition of key codes is not in the scope of the keyboard driver specification and has to be defined in the system. The process that uses the keyboard driver is responsible for mapping the key codes to application-specific information.

A default typematic rate (characters/sec) and typematic delay (msec) can be configured causing the driver to signal additional key press (repeat) events. The typematic rate and delay are part of the driver control block data type.

NOTE: The typematic rate functionality is not a basic requirement on the keyboard driver and therefore does not have to be implemented. This functionality should only be implemented if the hardware supports this feature.

2 Interface description of the KBD driver

2.1 Data types

Name	Description
kbd_DCB_Type	Driver Control Block

2.1.1 kbd_DCB_Type – Driver Control Block

Definition:

```
typedef struct kbd_DCB_Type
{
  USHORT       TypematicRate
  USHORT       TypematicDelay
}
```

Description:

The driver control block data type contains the typematic settings including the typematic rate in characters/sec and the typematic delay in msec. The driver can accept any values. The following table shows the set of default values defined.

NOTE: The typematic rate functionality is not a basic requirement on the keyboard driver and therefore does not have to be implemented. This functionality should only be implemented if it is supported by the hardware.

Type	Value (Char/Sec)
KBD_TYPEREATE_NONE	0
KBD_TYPERATE_6	6
KBD_TYPERATE_8	8
KBD_TYPERATE_10	10
KBD_TYPERATE_12	12
KBD_TYPERATE_15	15
KBD_TYPERATE_20	20
KBD_TYPERATE_24	24
KBD_TYPERATE_30	30

Type	Value (msec)
KBD_TYPEDELAY_250	250
KBD_TYPERATE_500	500
KBD_TYPERATE_750	750
KBD_TYPERATE_1000	1000

2.2 Constants

Name	Description
KBD_KEYDOWN	Indicates a key is pressed
KBD_KEYUP	Indicates a key is released
KBD_KEYREPEAT	Indicates auto repeat of a pressed key
KBD_SIGTYPE_STATUSCHG	Keyboard status change signal
KBD_TYPERATE_NONE	No automatic repetition of keys
KBD_TYPERATE_6	6 characters per second
KBD_TYPERATE_8	8 characters per second
KBD_TYPERATE_10	10 characters per second
KBD_TYPERATE_12	12 characters per second
KBD_TYPERATE_15	15 characters per second
KBD_TYPERATE_20	20 characters per second
KBD_TYPERATE_24	24 characters per second
KBD_TYPERATE_30	30 characters per second
KBD_TYPEDELAY_250	250 msec delay
KBD_TYPEDELAY_500	500 msec delay
KBD_TYPEDELAY_750	750 msec delay
KBD_TYPEDELAY_1000	1000 msec delay

2.3 Functions

Name	Description
kbd_Init	Initialization of the driver
kbd_Exit	Termination of the driver
kbd_SetConfig	Set typematic configuration
kbd_GetConfig	Retrieve typematic settings
kbd_SetSignal	Define a signal the driver uses to indicate an event
kbd_ResetSignal	Un-define a signal the driver uses to indicate an event
kbd_GetStatus	Retrieve the status of the keyboard

2.3.1 kbd_Init – Driver Initialization

Definition:

```
UBYTE kbd_Init
(
    drv_SignalCB_Type* in_SignalCBPtr
) ;
```

Parameters:

Name	Description
in_SignalCBPtr	This parameter points to the function that is called at the time an event that is to be signaled occurs. This value can be set to NULL if event signaling should not be possible.

Return values:

Name	Description
DRV_OK	Initialization successful
DRV_INITIALIZED	Driver already initialized
DRV_INITFAILURE	Initialization failed

Description

The function initializes the drivers internal data. The function returns DRV_INITIALIZED if the driver has already been initialized and is ready to be used or is already in use. In case of an initialization failure, which means that the driver cannot be used, the function returns DRV_INITFAILURE.

After initialization, the driver is ready to handle keyboard status changes.

2.3.2 kbd_Exit – De-initialization of the driver

Definition:

```
void kbd_Exit
(
    void
) ;
```

Parameters:

Name	Description
-	-

Return values:

Name	Description
-	-

Description

The function is called when the driver functionality is no longer needed. The function “de-allocates” all allocated resources and finalizes the driver.

2.3.3 kbd_SetConfig – Setup typematic configuration

Definition:

```
UBYTE kbd_SetConfig
(
    kbd_DCB_Type*          in_DCBPtr
) ;
```

Parameters:

Name	Description
in_DCBPtr	Pointer to the driver control block

Return values:

Name	Description
DRV_OK	Function successfully completed
DRV_INVALID_PARAMS	One or more values are out of range or invalid in that combination

Description

This function is used to set the typematic rate settings of the keyboard driver. After a successful completion, the driver uses the new configuration on following keyboard events (e.g. key press). Refer to Chapter 2.1.1 for more details about the driver control block.

If one of the parameters included in the driver control block is invalid, the function returns DRV_INVALID_PARAMS.

To retrieve the driver's default configuration, call the function kbd_GetConfig().

NOTE: The typematic rate functionality is not a basic requirement on the keyboard driver and therefore does not have to be implemented. This functionality should only be implemented if the hardware supports this feature.

2.3.4 kbd_GetConfig – Retrieve typematic configuration

Definition:

```
UBYTE kbd_GetConfig
(
    kbd_DCB_Type*         out_DCBPtr
) ;
```

Parameters:

Name	Description
out_DCBPtr	Pointer to the driver control block

Return values:

Name	Description
DRV_OK	Function successfully completed
DRV_DRIVER_NOTCONFIGURED	The driver is not yet configured

Description

This function is used to retrieve the typematic rate settings of the driver. The configuration is returned in the driver control block to which the pointer provided out_DCBPtr points. The typematic configuration can be set by using the kbd_SetConfig() function. Refer to Chapter 2.1.1 for more details about the driver control block.

If the driver is not configured, the function returns DRV_NOTCONFIGURED.

NOTE: The typematic rate functionality is not a basic requirement on the keyboard driver and therefore does not have to be implemented. This functionality should only be implemented if the hardware supports this feature.

2.3.5 kbd_SetSignal – Setup a Signal

Definition:

```
UBYTE kbd_SetSignal
(
        drv_SignalID_Type*      in_SignalIDPtr
) ;
```

Parameters:

Name	Description
in_SignalIDPtr	Pointer to the signal information data

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function is used to define a signal that indicates keyboard status changes to the process. A keyboard status change is an event identified in the signal information data type as SignalType. The only signal that can be set is the keyboard status change signal defined in the following table.

The parameter UserData which is part of the SignalID data type is ignored when calling kbd_SetSignal().

Signal	Value
KBD_SIGTYPE_STATUSCHG	DRV_SIGTYPE_USER

Figure 1

The parameter UserData is only valid at the time the driver calls the specified signal call-back function. In this case, the contents of the UserData parameter has the structure shown in Figure 2 and is named “key-status”. For more information about the data type drv_SignalID_Type, refer to the document “Generic Driver Interface”.

Parameter	Position	Contents
UserData (key-status)	Lo-Word (Bits 0 – 15)	KeyCode
	Hi-Word (Bits 16 – 31)	KeyStatus

Figure 2

The KeyCode is system and implementation dependent. The process which is using the keyboard driver is responsible for mapping the key codes to application specific information. The key status, which is the Hi-Word of the UserData parameter represents the status of the key. Possible values are shown in Figure 3.

Key Status	Value	Description
KBD_KEYDOWN	1	The key is pressed
KBD_KEYUP	2	The key is released
KBD_KEYREPEAT	3	The key is pressed and because of the typematic rate settings it is simulated that the key has been released and then pressed again

Figure 3

To remove a signal, call the function kbd_ResetSignal().

If one of the parameters of the signal information data is invalid, the function returns DRV_INVALID_PARARMS.

If no signal call-back function has been defined at the time of initialization, the driver returns DRV_SIGFCT_NOTAVAILABLE.

2.3.6 kbd_ResetSignal – Remove a Signal

Definition:

```
UBYTE kbd_ResetSignal
(
        drv_SignalID_Type*      in_SignalIDPtr
) ;
```

Parameters:

Name	Description
in_SignalIDPtr	Pointer to the signal information data

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function is used to remove a signal that has previously been set. The signal that is removed is identified by the Signal Information Data element called SignalType. All other elements of the Signal Information Data must be identical to the signal that is to be removed (process handle and signal value). For more information about the data type drv_SignalID_Type, refer to the document "Generic Driver Interface".

If no signal call-back function has been defined at the time of initialization, the driver returns DRV_SIGFCT_NOTAVAILABLE.

2.3.7 kbd_GetStatus – Retrieve the keyboard status

Definition:

```
ULONG kbd_GetStatus
(
        void
) ;
```

Parameters:

Name	Description
-	-

Return values:

Name	Description
Keystatus	Current status of the keyboard

Description

This function is used to retrieve the current (latest) status of the keyboard. The return value is structured as shown in Figure 2.

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/>