Technical Document - Confidential

GSM PROTOCOL STACK

G23

EMI-ELECTRICAL MAN-MACHINE INTERFACE

DRIVER INTERFACE

Document Number:	8415.003.99.008
Version:	0.9
Status:	Draft
Approval Authority:	
Creation Date:	1998-Sep-2
Last changed:	2015-Mar-08 by XGUTTEFE
File Name:	8415_003.doc

Important Notice

Change History

Date	Changed by	Approved by	Version	Status	Notes
1998-Sep-2	LM et al.		0.1		1
1998-Sep-9	LM et al.		0.2		2
1998-Nov-3	LM et al.		0.3		3
1999-Mar-9	LM et al.		0.4		4
1999-Mar-18	MS et al.		0.5		5
1999-Sep-6	LM et al.		0.6		6
1999-Nov-16	FR et al		0.7		7
2000-Jan-18	MP et al.		0.8		8

2003-May-13	XINTEGRA		0.9		

Notes:

1. Initial version
2. Now complies with [C_8415.0026]
3. API changed
4. Now complies with new 8415.026.99.012 version
5. Format/English check
6. API changed. Now complies with socket API
7. API changed again to fit in test interface layer struct configuration via string; some extra explanations
8. Adaption to new GDI

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

1.1 References

[1]	GSM 4.14, Individual equipment type requirements and interworking; Special conformance testing functions TS 101 293 V6.0.0, ETSI, April 1999
[2]	Generic Driver Interface – Functional Specification; Condat 8415.026.99.012; March 19, 1999
[3]	VSI/PEI – Frame/Body Interfaces - Next Generation; Condat 8415.033.99.100
[4]	Concept for Integration of Drivers into the Test Interface; Condat internal, Nov 99

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 [02.30] and
- Test and integration support tools.

This document describes the functional interface of the G23 Layer 2 test interface driver, EMI. The EMMI functionality was originally defined by GSM in the recommendation 11.10, later moved do 4.14 [1]. This functionality has been extended to fulfill the needs of the G23 Protocol Stack, including higher data rates and larger data frames than the original EMMI.

This driver allows secure data transmission and reception. Layer 1 may be implemented by any driver that fits the test interface driver structure, e.g. USART or SOCKET.

In addition, this implementation of the EMI Layer 2 protocol includes a watchdog functionality. This means that the driver is able to send heartbeats (XONs) to the remote end to indicate the active and ready to receive state. If the remote end does not send any data or heartbeats, the driver indicates the status DRV_SIGTYPE_DISCONNECT.

2.1 G23 Test Interface Layer 2

In Layer 2, frames are used to carry data from higher layers via Layer 1. Special Layer 2 frames (control frames) are used to control the flow and allow a secure transmission of the frames.

The flow control used by the G23 Test Interface Layer 2 is in line with the flow control used by EMMI. A second kind of I frame has been introduced that can carry more data in a single frame.

The following tables show the structure of both I frames.

Field name	# of octets	Value	Start at octet #
Start	1	Character STX	1
Length	1	Length of data	2
Data	0-255	Content of data	3
Check	1	Error detection	Length + 3
End	1	Character ETX	Length + 4

Table 1: Original I frame

Field name	# of octets	Value	Start at octet #
Start	1	Character SI	1
Length	2	Length of data	2
Data	0-65535	Content of data	4
Check	1	Error detection	Length + 4
End	1	Character ETX	Length + 5

Table 1: Extended I Frame

The EMI Driver accepts both frames in receiving. It sends the original frame, if the data length is lower than 256, else the extended frame.

The retransmission of I frames by the G23 Test Interface Layer 2 is limited to a maximum of 3 re-transmissions. If the I frame could not be transmitted it will be discarded and the higher layer is notified about the failed retransmission.

The EMI driver described in this document implements this Layer 2.

2.2 G23 Test Interface Layer 1

Layer 1 is performed by another driver, which is called by EMI. The functional interface of this driver has to be the same in structure as it is for EMI. The design of the test interface driver structure is explained in more detail in [4].

3 Interface description of the EMI driver

3.1 Data Types

For types not defined here see [2]

3.1.1 T_DRV_FUNC – Diver Functions

Definition:

```
typedef struct
{
  void (*drv_Exit)();
  USHORT (*drv_Read)();
  USHORT (*drv_Write)();
  USHORT (*drv_Look)();
USHORT (*drv_Clear)();
  USHORT (*drv_Flush)();
  USHORT (*drv_SetSignal)();
  USHORT (*drv_ResetSignal)();
  USHORT (*drv_SetConfig)();
  USHORT (*drv_GetConfig)();
  void (*drv_Callback)();
} T_DRV_FUNC
```

Description: The structure of the type T_DRV_FUNC contains the addresses of the driver entry functions.

3.1.2 T_DRV_EXPORT – Driver Properties

Definition:

```
typedef struct
{
  char *            Name
  USHORT            Flags
  T_DRV_FUNC        DrvFunc
} T_DRV_EXPORT
```

Description:

This data type defines the properties exported by the driver.

Name	Name of the driver
Flags	Bit (0): Callback function is called during ISR(1)/not called during ISR(0)
DrvFunc	functions to access the driver

3.1.3 T_DRV_SIGNAL – Driver Signal

Definition:

```
typedef struct
{
        USHORT SignalType
        USHORT DataLength
        void *      UserData
        USHORT DrvHandle
} T_DRV_SIGNAL
```

Description:

This type defines the signal information data used to identify a signal. This data type is used to define and to report a signal. A signal is defined by a process by calling the driver function *drv*_SetSignal(). An event is signaled by a driver by calling the signal call-back function (see 3.1.4, 3.3, 3.4.1).

3.1.4 T_DRV_CB_FUNC – Driver Callback Function

Definition:

```
typedef void (*T_DRV_CB_FUNC) (T_DRV_SIGNAL * Signal) ;
```

Description:

This type defines a call-back function used to signal driver events, e.g. driver is ready to accept data. The driver calls the signal call-back function when a specific event occurs and the driver has been instructed to signal the event to a specific process.

A process can set or reset event signaling by calling one of the driver functions emi_SetSignal(), emi_ResetSignal(). Event signaling can only be performed when a signal call-back function has been installed at driver initialization.

For more information about the T_DRV_SIGNAL data type, refer to 3.1.3.

3.2 Constants

3.2.1 Return Codes

Name	Description
DRV_OK	Return value indicating the function completed successfully
DRV_INPROCESS	The requested function is currently being executed
DRV_NOTCONFIGURED	Driver is not configured
DRV_INITFAILURE	Driver initialization failed
DRV_BUFFER_FULL	The internal buffer is exhausted
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid
DRV_NOCONNECT	No connection exists to a peer entity
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available
DRV_ERROR	Other error

3.2.2 Signal Codes

Name	Description
DRV_SIGTYPE_READ	Used to specify read event signaling
DRV_SIGTYPE_WRITE	Used to specify write event signaling
DRV_SIGTYPE_CONNECT	Used to specify connection established event signaling
DRV_SIGTYPE_DISCONNECT	Used to specify connection release event signaling

3.3 Signals

Signals are used to asynchronously inform the process using services about selected events. A signal call-back function is passed to the driver at the time of initialization (see “3.4.1 emi_Init – Driver Initialization”). When no call-back is defined, event signaling cannot be performed. A signal can be set using the function emi_SetSignal(), which can be found in Chapter 3.4.7. Event signaling can be disabled by calling the function emi_ResetSignal().

The signalling is actually performed by calling the callback function, which has the following proto type:

```
void callback( T_DRV_SIGNAL *Signal );
```

The EMI driver uses only the *DrvHandle* and *SignalType* of the Signal, *UserData* and *DataLength* are nor used. The following table shows the values for *SignalType* as used by EMI:

SignalType	**meaning**
DRV_SIGTYPE_READ	The driver has received data which can be read by the emi_Read() function.
DRV_SIGTYPE_WRITE	The driver has written data so that the write buffer is ready to take new data (using emi_Write()).
DRV_SIGTYPE_CONNECT	EMI detected that the peer entity is working, i.e. there is a connection. Data may be sent.
DRV_SIGTYPE_DISCONNECT	EMI detected that the connection to the peer entity is lost.

3.4 Functions

Name	Description
emi_Init	Initialization of EMI
emi_Exit	Termination of EMI
emi_Write	Send data to the remote end
emi_Read	Read data from the driver
emi_Clear	Clear internal buffers
emi_SetSignal	Define a signal which the driver uses to indicate an event
emi_ResetSignal	Un-define a signal which the driver uses to indicate an event
emi_SetConfig	Set driver configuration
emi_GetConfig	Get driver configuration
emi_Callback	Callback entry for layer 1 driver

The GDI standard functions emi_Look() and emi_Flush() are not implemented.

3.4.1 emi_Init – Driver Initialization

Definition:

```
USHORT emi_Init
(
    USHORT              DrvHandle,
    T_DRV_CB_FUNC       CallbackFunc,
    T_DRV_EXPORT **     DrvInfo
) ;
```

Parameters:

Name	Description
DrvHandle	This is the number, which identifies the driver in the system. The driver has to store it. It is needed, when the callback function is called.
CallbackFunc	The address of the callback function. The events at which to call back are configured using emi_SetSignal(). If this address is equal to NULL the driver will not try to call back.
DrvInfo	Via this parameter the driver returns the adresses of it's functions. Some function addresses may be set to NULL, which means that these functions are not implemented and cannot be called.

Return values:

Name	Description
DRV_OK	Initialization successful
DRV_INITFAILURE	Initialization failed

Description

This function initializes the driver's internal data and returns the addresses of it's entry functions. emi_Init() returns DRV_OK in the case of a successful completion.

In case of an initialization failure, which means that the driver cannot be used, the function returns DRV_INITFAILURE.

All signals have to be disabled during initialization and may be enabled later.

The next call to EMI after emi_Init() shall be emi_SetConfig() !

3.4.2 emi_Exit – Driver Finalization

Definition:

```
void emi_Exit
(
    void
) ;
```

Parameters:

Name	Description
-	

Return values:

Name	Description
-	-

Description

The function is called when the driver functionality is no longer required. The data exchange terminates immediately regardless of any outstanding data to be sent.

3.4.5 emi_Read - Read Data from the Driver

Definition:

```
USHORT emi_Read
(
        void *          Buffer,
        USHORT *        Length
) ;
```

Parameters:

Name	Description
Buffer	This parameter points to the buffer wherein the data is to be copied
Length	On call: number of characters to read. If the function returns DRV_OK, it contains the number of characters read. If the function returns DRV_INPROCESS, it contains 0.

Return values:

Name	Description
DRV_OK	Function successful
DRV_INPROCESS	The driver is currently reading data. The data is incomplete.
DRV_NOTCONFIGURED	The driver is not yet configured
DRV_NOCONNECT	Connection not available

Description

This function is used to read one data frame from the driver. The data is copied into the buffer to where Buffer points. The parameter *Length contains the size of the buffer in characters.

If the driver is not configured, the function returns DRV_ NOTCONFIGURED.

If no connection exists the driver returns DRV_NOCONNECT.

NOTE: When calling the function with a buffer size of 0, the function will return DRV_OK. The size of the buffer needed to store the available data is stored in the parameter *Length. In this case, Buffer can be set to NULL.

3.4.6 emi_Write – Write Data to the Driver

Definition:

```
USHORT emi_Write
(
        void *              Buffer,
        USHORT *            Length
) ;
```

Parameters:

Name	Description
Buffer	This parameter points to the buffer that is passed to the driver for further processing
Length	On call: number of characters to write. If the function returns DRV_OK, it contains the number of characters written. This number is always equal to the requested number. In all other cases it contains 0.

Return values:

Name	Description
DRV_OK	Function successful, data frame written
DRV_BUFFER_FULL	Not enough space
DRV_NOTCONFIGURED	The driver is not yet configured
DRV_NOCONNECT	Connection not available

Description

This function is used to write one data frame to the driver. The parameter *Length contains the number of characters to write.

In the case of a successful completion, the function returns DRV_OK.

In all other cases the data was not written and the call to emi_Write() should be repeated later.

The emi_Write() function has to call vsi_d_write() – see prototype below - to call the layer 1 *drv*_Write() function.

```
USHORT vsi_d_write ( USHORT Caller, USHORT DrvHandle, void *Buffer, USHORT *Length
)
```

The parameter Caller is the handle of the EMI driver passed to emi_Init(). The parameter DrvHandle has to be set to 0 to indicate to the frame that it must call the corresponding lower layer *drv*_Write() function. Buffer and Length are the parameters passed to emi_Write(). For detailed information about vsi_d_write() refer to [3].

NOTE: When calling the function with a buffer size of 0, the function will return the number of characters that can be written in the parameter *Length. In this case, Buffer can be set to NULL, no data will be written.

3.4.7 emi_SetSignal – Setup a Signal

Definition:

```
USHORT emi_SetSignal
(
        USHORT                    SignalType
) ;
```

Parameters:

Name	Description
SignalType	Signal type to be set

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	Signal type not supported by EMI
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function is used to define a single signal that the driver should indicated via the callback function.

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

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

3.4.8 emi_ResetSignal – Remove a Signal

Definition:

```
USHORT emi_ResetSignal
(
        USHORT                  SignalType
) ;
```

Parameters:

Name	Description
SignalType	Signal type to be reset

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	Signal type not supported by EMI
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function is used to remove a previously set signal.

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

3.4.9 emi_SetConfig – Set the Driver Configuration

Definition:

```
USHORT emi_SetConfig
(
    char *                   Config
) ;
```

Parameters:

Name	Description
Config	Pointer to driver configuration string

Return values:

Name	Description
DRV_OK	Function successfully completed
DRV_INVALID_PARAMS	Error in configuration parameters

Description

This function is used to configure the driver and must be called after emi_Init() but before any other function.

3.4.10 emi_GetConfig – Retrieve the Driver Configuration

Definition:

```
USHORT emi_GetConfig
(
    char *                 Buffer
) ;
```

Parameters:

Name	Description
Buffer	Address to copy the configuration data string to.

Return values:

Name	Description
DRV_OK	Function successfully completed
DRV_INVALID_PARAMS	Driver not configured

Description

This function is used retrieve the configuration of the driver. The configuration data is copied to the Buffer provided by the caller.

If the driver has not been configured DRV_NOTCONFIGURED is returned.

3.4.11 emi_Callback – Callback Function

Definition:

```
void emi_Callback
(
    T_DRV_SIGNAL *        Signal
) ;
```

Parameters:

Name	Description
Signal	Pointer to Signal sent by layer 1 driver

Return values:

Name	Description
-	

Description

This function is called by the frame if the lower layer driver calls the vsi_d_callback() function.

If the signal type in the passed signal is DRV_SIGTYPE_READ the emi_Callback() function has to call vsi_d_read() – see prototype below - to read the layer 1 data via its *drv*_Read() function.

```
USHORT vsi_d_read ( USHORT Caller, USHORT DrvHandle, void *Buffer, USHORT *Length )
```

The parameter Caller is the handle of the EMI driver passed to emi_Init(). The parameter DrvHandle has to be set to 0 to indicate to the frame that it must call the corresponding lower layer *drv*_Read() function. For detailed information about vsi_d_read() refer to [3].

The data read from the lower layer driver has to be analyzed and when a complete message frame has been received and the DRV_SIGTYPE_READ is set then the callback function, which address was passed to emi_Init(), has to be called with a pointer to a signal of the type T_DRV_SIGNAL (refer to 3.1.3). The signal type has to be set to DRV_SIGTYPE_READ and the DrvHandle is the one passed to emi_Init().

Signal types different from DRV_SIGTYPE_READ like DRV_SIGTYPE_CONNECT and DRV_SIGTYPE_DISCONNECT also have to be forwarded to the upper layer driver - if the corresponding signal type is enabled - by calling the callback function, which address was passed to emi_Init().

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/>