Technical Document

GSM PROTOCOL STACK

G23

GDI – GENERIC DRIVER INTERFACE

FUNCTIONAL SPECIFICATION

Document Number:	8415.026.99.020
Version:	0.18
Status:	Draft
Approval Authority:	
Creation Date:	1998-Sep-08
Last changed:	2015-Mar-08 by Joerg Deiss
File Name:	8415_026.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
1998-Sep-08	LM et al.		0.1		1
1998-Oct-28	LM et al.		0.2		2
1998-Nov-24	MS et al.		0.3		3
1998-Dec-14	LM et al.		0.4		4
1998-Dec-15	LM et al.		0.5		5
1998-Dec-17	LM et al.		0.6		6
1999-Mar-02	MS et al.		0.7		7
1999-Mar-10	LM et al.		0.8		8

1999-Jun-04	LE et al.		0.9		9
1999-Dec-01	TSE et al.		0.10		10
1999-Dec-08	TSE et al.		0.11		11
2000-Jan-11	MP et al.		0.12		12
2000-Jan-19	MP et al.		0.13		13
2000-Jan-19	MP et al.		0.14		14
2000-Jun-07	MP		0.15		15
2001-Oct-14	SBK		0.16		16
2001-Nov-13	MP		0.17		17

Notes:

1. Initial version
2. API changed
3. Editorial
4. Complete API change
5. Editorial, auto data types changed
6. Draft version
7. New document template/English check
8. Introduction of the chapter “Signals”/Clarification of the use of ProcHandle
9. Consistency check/Submitted
10. Introduction, variable type updated
11. Drv_SignalID_Type updated
12. Redesign of data types and functions
13. T_DRV_SIGNAL modified
14. T_DRV_SIGNAL modified
15. Update the current gdi.h
16. Editorial improvements, minor corrections
17. Change length values to ULONG

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

[C_8415.033]	8415.033, VSI/PEI- Frame/Body Interfaces; Condat

1.2 Abbreviations

API	Application Programming Interface
GDI	Generic Driver Interface
NVRAM	Non Volatile Random Access Memory

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/AT Command Interface,
- MMI and MMI Framework (MFW) and
- Test and integration support tools.

This document describes the general design of the drivers and driver interfaces (driver APIs) used in the G23 Protocol Stack. Drivers are function libraries that export reusable functionality for different applications (processes). In general, drivers do not attend to scheduling.

There are two kinds of drivers recognized in the context of the G23 Protocol Stack, communication drivers and "plain" function drivers.

Communication drivers include communication functionality. This means that they can send data to and receive data from a hardware device such as a serial interface. The communication drivers signal specific events, such as the reception of data, to the parent application using one of two methods (signals or call-backs). Signals are part of the common driver API specification. Call-back functionality is an optional implementation and is therefore not described in this document.

In contrast to communication drivers, plain drivers act as simple common function libraries that can access hardware devices. An LED driver, or a ring buffer manager are examples for such a plain drivers. They allow processes to write in and read from the NVRAM via one standardized interface, the driver.

The following chapters describe the functions from which the drivers must be derived. There is also a differentiation between plain drivers and communication drivers, in that a different amount of standard functions must be implemented. All drivers must implement the functions *drv*_Init() and *drv*_Exit(). All communication drivers must implement the functions *drv*_Read(), *drv*_Write(), *drv*_Look(), *drv*_Flush(), *drv*_SetSignal(), *drv*_ResetSignal() and *drv*_Clear(). Drivers may implement the functions *drv*_SetConfig(), *drv*_GetConfig() and *drv*_Callback().

function	**plain driver**	**communication driver**
*drv*_Init()	mandatory	mandatory
*drv*_Exit()	mandatory	mandatory
*drv*_Read()	optional	mandatory
*drv*_Write()	optional	mandatory
*drv*_Look()	optional	mandatory
*drv*_Flush()	optional	mandatory
*drv*_SetSignal()	optional	mandatory
*drv*_ResetSignal()	optional	mandatory
*drv*_Clear()	optional	mandatory
*drv*_SetConfig()	optional	optional
*drv*_GetConfig()	optional	optional
*drv*_Callback()	optional	optional

drv signifies the name of the specific driver e.g. emi_Init(). Instances of *drv* will have to be substituted by a sequence of letters denoting the actual driver.

The standard driver API as defined by this document may be extended by adding additional functions, constants and data types.

3 Generic Driver Interface

3.1 Data types

Name	Description
ULONG	Unsigned 32 bit integer data type
USHORT	Unsigned 16 bit integer data type
SHORT	Signed 16 bit integer data type
T_VOID_STRUCT	Unsigned 32 bit integer data type (equivalent to unsigned long)
T_*DRV*_DCB	Driver Control Block (may be different for different drivers)
T_DRV_SIGNAL	Signal information data type
T_DRV_CB_FUNC	Signal call-back function type
T_DRV_FUNC	Driver's function type
T_DRV_LIST	Drivers List data type
T_DRV_EXPORT	Driver's property type (exported by the driver)

3.1.1 T_*DRV*_DCB – Driver Control Block

Definition:

```
typedef struct T_DRV_DCB
{
  …
 } T_DRV_DCB;
```

Description:

The driver control block data type T_*DRV*_DCB is a prototype for driver-specific implementations of a driver control block. A driver control block contains all parameters used to configure a driver.

3.1.2 T_DRV_CB_FUNC – Driver Call-Back 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 (see 3.5.8).

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

The signal call-back has only one single parameter Signal containing all data required to identify the signal. For more information about the T_DRV_SIGNAL data type, refer to 3.1.3.

3.1.3 T_DRV_SIGNAL – Driver Signal

Definition:

```
typedef struct
{
  USHORT           SignalType;
  USHORT           DrvHandle;
  ULONG            DataLength;
  T_VOID_STRUCT *  UserData;
} 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 pre-defined signal call-back function (see 3.5.1).

3.1.4 T_DRV_FUNC – Driver 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:

This type defines the functions exported by the driver. If any of the functions is not implemented for a driver, a NULL-pointer has to be entered in this table.

3.1.5 T_DRV_LIST_ENTRY – Driver List

Definition:

```
typedef struct
{
  char *           Name;
  USHORT           (*drv_Init)( USHORT, T_DRV_CB_FUNC, T_DRV_EXPORT const **);
  char *           Process;
  void *           DrvConfig;
} T_DRV_LIST_ENTRY
```

Description:

This data type defines the parameters needed to setup a driver.

Name name of the driver

drv_Init() driver initialization function

Process process to be notified by driver call-back, e.g. "TST"

DrvConfig pointer to driver control block

3.1.6 T_DRV_EXPORT – Driver's Properties (exported by the Driver)

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): Call-back function is called during ISR(1)/not called during ISR(0)

DrvFunc functions to access the driver

3.2 Constants

Name	Description
DRV_BUFFER_FULL	The internal buffer is exhausted
DRV_DISABLED	Driver is not enabled
DRV_ENABLED	Driver is enabled
DRV_NOTCONFIGURED	Driver is not configured
DRV_INITFAILURE	Driver initialization failed
DRV_INITIALIZED	Driver is already initialized
DRV_INTERNAL_ERROR	Unspecified internal driver error
DRV_INPROCESS	The requested function is currently being executed
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid
DRV_OK	Return value indicating the function completed successfully
DRV_SIGFCT_NOTAVAILABLE	The requested event signaling functionality is not available
DRV_SIGTYPE_CLEAR	Used to specify clear ready event signaling (refer to 3.5.8)
DRV_SIGTYPE_FLUSH	Used to specify flush ready event signaling (refer to 3.5.8)
DRV_SIGTYPE_READ	Used to specify read event signaling (refer to 3.5.8)
DRV_SIGTYPE_USER	Used to specify additional driver dependent signals (refer to 3.5.8)
DRV_SIGTYPE_WRITE	Used to specify write event signaling (refer to 3.5.8)
DRV_BUFTYPE_WRITE	Used to specify the write buffer type of buffer (refer to 3.5.6)
DRV_BUFTYPE_READ	Used to specify the read buffer type of buffer (refer to 3.5.6)
DRV_MAX_SIGNAL	Maximum number of processes to be notified by a driver call-back

3.3 Signals

Signals are used to inform the using process about selected events asynchronously. Signaling is done by passing a signal call-back function to the driver at the time of initialization (see "3.5.1 *drv_Init* – Driver Initialization"). When no call-back function is defined event signaling cannot be performed. A signal can be set using the function *drv*_SetSignal() which can be found in Chapter 3.5.8. Event signaling can be disabled by calling the function *drv*_ResetSignal(), for more details on this function refer to Chapter 3.5.9.

The following chapters describe the contents of the T_DRV_SIGNAL information structures defined for the common driver signals.

3.3.1 DRV_SIGTYPE_READ

This signal is indicated when the driver has received data which can now be read using the *drv*_Read() function. A prerequisite to being informed asynchronously about this event is that the signal has been set using the *drv*_SetSignal() function. The event will only be signaled each time new data is available. The behavior depends on the driver's functionality i.e. the event may be signaled for example each time a character is received via an RX line of a RS232 HW or each time a complete data block is available. Call the *drv*_Look() or *drv*_Read() function as a reaction to the signal.

Parameter	**Value**
SignalType	DRV_SIGTYPE_READ
DataLength	not used
UserData	not used

3.3.2 DRV_SIGTYPE_WRITE

This signal will be indicated when the write buffer is ready to take new data. A prerequisite to being informed asynchronously about this event is that the signal has been set using the *drv*_SetSignal() function. The event will only be signaled once, however each time the write buffer is ready to take new data using the *drv*_Write() function.

Parameter	**Value**
SignalType	DRV_SIGTYPE_WRITE
DataLength	not used
UserData	not used

3.3.3 DRV_SIGTYPE_CLEAR

This signal is indicated when the read and/or write buffer of the driver has been cleared asynchronously. Prerequisite to being informed asynchronously about this event is that the signal has been set using the *drv*_SetSignal() function and a call to the *drv*_Clear() function has been performed which returned DRV_INPROCESS meaning that the buffers could not be cleared at once (synchronously) or clearing is currently in process. The event will be signaled as soon as the selected buffer(s) is/are cleared.

Parameter	**Value**
SignalType	DRV_SIGTYPE_CLEAR
DataLength	sizeof(USHORT)
UserData	Buffertype (Bitmask, values: DRV_BUFTYPE_WRITE, DRV_BUFTYPE_READ)

3.3.4 DRV_SIGTYPE_FLUSH

This signal is indicated when the driver buffer has been flushed asynchronously. Prerequisite to being informed asynchronously about this event is that the signal has been set using the *drv*_SetSignal() function and that a call to the *drv*_Flush() function has been performed which returned DRV_INPROCESS meaning that the buffers could not be flushed at once (synchronously) or flushing is currently being performed.

Parameter	Value
SignalType	DRV_SIGTYPE_FLUSH
DataLength	not used
UserData	not used

3.4 Driver Setup and Call-back Mechanism

All test interface drivers used in the protocol stack are entered in a driver list. In this list the address of the *drv*_Init() function, the names of the entitiy, that must be notified in the case of a driver call-back, and a pointer to a default configuration string.

The frame reads the driver list, calls the *drv*_Init() function and stores the handles of the process to be in a table under the index corresponding to the index in the driver list. This index is also passed to the *drv*_Init() function and serves as the driver handle. The address of a call-back function that is located in the frame is also passed to the *drv*_Init() function and a pointer to the properties exported by the driver is stored in the driver table.

In the case of a test interface driver call-back, the frame finally notifies the process which name is entered for the uppermost driver in the driver list and passes a signal to it if the corresponding signal type is enabled. The frame acts as dispatcher.

To enable/disable signal types the functions vsi_d_setsignal() or vsi_d_resetsignal() must be used, refer to [C_8415.033].

3.5 Functions

Name	Description
*drv*_Init	Initialization of the driver
*drv*_Exit	Termination of the driver
*drv*_Clear	Re-initialize all buffers
*drv*_Write	Write data to the driver
*drv*_Look	Read data from a driver but leave data unchanged
*drv*_Read	Read data from the driver
*drv*_Flush	Flush all buffers
*drv*_SetSignal	Define a signal the driver uses to indicate an event
*drv*_ResetSignal	Un-define a signal the driver uses to indicate an event
*drv*_SetConfig	Set driver configuration
*drv*_GetConfig	Get driver configuration
*drv*_Callback	Callback function of the higher layer driver

3.5.1 *drv*_Init – Driver Initialization

Definition:

```
USHORT drv_Init
(
    USHORT              DrvHandle
    T_DRV_CB_FUNC       CallbackFunc
    T_DRV_EXPORT **     DrvInfo
) ;
```

Parameters:

Name	Description
DrvHandle	unique handle of the driver
CallbackFunc	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. This function must not be confused with the *drv*_Callback() function in 3.5.12
DrvInfo	Pointer to the driver parameters.

Return values:

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

Description

This function needs to be implemented in all drivers.

The driver exports its properties like its name, the functions to access driver functionality and a bitfield called flags. As the drivers exports only the address of its properties, these must not be located on the stack but as static structure.

The function initializes the driver's internal data. The function returns DRV_OK in the case of a successful completion.

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.

3.5.2 *drv*_Exit – Driver Finalization

Definition:

```
void drv_Exit
(
    void
) ;
```

Parameters:

Name	Description
-	-

Return values:

Name	Description
-	-

Description

This function needs to be implemented in all drivers.

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

3.5.3 *drv*_Read - Read Data from the Driver

Definition:

```
USHORT drv_Read
(
        void *              Buffer
        ULONG *             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.

Description

This function needs to be implemented in all communication drivers.

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

In the case of a successful completion, the driver's buffer is cleared. The driver keeps the data available when calling the function drv_Look().

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.

3.5.4 *drv*_Write – Write Data to the Driver

Definition:

```
USHORT drv_Write
(
        void *              Buffer
        ULONG *             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_BUFFER_FULL, it contains the maximum number of characters that can be written. If the function returns DRV_OK, it contains the number of characters written. If the function returns DRV_INPROCESS, it contains 0.

Return values:

Name	Description
DRV_OK	Function successful
DRV_BUFFER_FULL	Not enough space
DRV_INPROCESS	Driver is busy writing data

Description

This function needs to be implemented in all communication drivers.

This function is used to write data 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.

If the data cannot be written because the storage capacity of the driver has been exhausted, the function returns DRV_BUFFER_FULL and the maximum number of characters that can be written in * Length.

If the driver is currently busy writing data and therefore cannot accept further data to be written, it returns DRV_INPROCESS and sets the parameter *Length to 0.

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.

3.5.5 *drv*_Look – Look at Data from the Driver

Definition:

```
USHORT drv_Look
(
        void *              Buffer
        ULONG *             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.

Description

This function needs to be implemented in all communication drivers.

This function is used to read data from the driver. The data is copied into the buffer to which Buffer points. The parameter *Length contains the size of the buffer in characters. The driver's internal buffer is not cleared.

In the case of a successful completion, the function returns DRV_OK and sets the value of *Length to the number of characters read.

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.

3.5.6 *drv*_Clear – Clear internal Driver Buffers

Definition:

```
USHORT drv_Clear
(
        USHORT              BufferType
) ;
```

Parameters:

Name	Description
BufferType	Bit-mask used to specify if the read, write or read and write buffer is cleared

Return values:

Name	Description
DRV_OK	Function successful
DRV_INPROCESS	The driver could not complete the clearance of the buffers at once. The driver is busy clearing the buffers.

Description

This function needs to be implemented in all communication drivers.

This function is used to clear the driver's internal buffers. The parameter BufferType is used to specify which buffer is to be cleared. The value of BufferType can be one of the values or a combination of the values defined in the following table. Combining DRV_BUFTYPE_READ and DRV_BUFTYPE_WRITE using bitwise OR will cause the driver to clear the read and write buffers.

Buffer type	Value
DRV_BUFTYPE_WRITE	1
DRV_BUFTYPE_READ	2

Figure 1

3.5.7 *drv*_Flush – Flush internal Driver Buffers

Definition:

```
USHORT drv_Flush
(
        void
) ;
```

Parameters:

Name	**Description**
-	-

Return values:

Name	**Description**
DRV_OK	Function successful
DRV_INPROCESS	The driver could not complete flushing the buffers at once. The driver is busy flushing the buffers.

Description

This function needs to be implemented in all communication drivers.

This function is used to flush the driver's internal buffers. This means data that is currently stored in the driver's internal write buffer is written to the device at once.

3.5.8 *drv*_SetSignal – Setup a Signal

Definition:

USHORT *drv*_SetSignal
(
        USHORT   SignalType
) ;

Parameters:

Name	Description
SignalType	Signal type to be set

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	Invalid signal type
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function needs to be implemented in all communication drivers.

This function is used to define a single signal or multiple signals that is/are indicated to the process when the event identified by SignalType occurs. Standard signals are defined in the following table. The signals can be extended for derived drivers, using DRV_SIGTYPE_USER shifted left to define new signals.

Signal	Value
DRV_SIGTYPE_WRITE	0x0001
DRV_SIGTYPE_READ	0x0002
DRV_SIGTYPE_FLUSH	0x0004
DRV_SIGTYPE_CLEAR	0x0008
DRV_SIGTYPE_USER	0x0010

Figure 2

To remove a signal, call the function *drv*_ResetSignal().

If the passed signal type is not implemented, the driver returns DRV_INVALID_PARAMS.

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

This function is normally not called directly by the process, but indirectly via the frame function vsi_d_setsignal(): For dynamic enabling of signals, the process calls vsi_d_setsignal() to enable the frame to store the new signal type mask for the calling process (refer to 3.4).

3.5.9 *drv*_ResetSignal – Remove a Signal

Definition:

```
USHORT drv_ResetSignal
(
        USHORT                SignalType
) ;
```

Parameters:

Name	Description
SignalType	Signal type to be reset

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	Invalid signal type
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function needs to be implemented in all communication drivers.

It is used to remove a single signal or multiple signals that have previously been set. The signal type is a bit mask containing the signal(s) to be reset.

If the passed signal type is not implemented the driver returns DRV_INVALID_PARAMS.

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

This function is normally not called directly by the process but indirectly via the frame function vsi_d_resetsignal(): For dynamic disabling of signals the process calls vsi_d_resetsignal() to enable the frame to store the new signal type mask for the calling process (refer to 3.4).

3.5.10 *drv*_SetConfig – Set a Driver Configuration

Definition:

```
USHORT drv_SetConfig
(
    T_DRV_DCB *            DCBPtr
) ;
```

Parameters:

Name	Description
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

Implementation of this function is optional for all types of drivers and governed by the functional specification of the respective driver.

This function is used to configure a driver (transmission rate, flow control, etc). For detailed information about the contents of the driver control block, refer to the functional specification of the specific driver.

If any value of the configuration is out of range or invalid in combination with any other value of the configuration, the function returns DRV_INVALID_PARAMS.

Call the *drv*_GetConfig() function to retrieve the driver's current configuration.

3.5.11 *drv*_GetConfig – Retrieve the Driver Configuration

Definition:

```
USHORT drv_GetConfig
(
    T_DRV_DCB *           DCBPtr
) ;
```

Parameters:

Name	Description
DCBPtr	Pointer to the driver control block

Return values:

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

Description

Implementation of this function is optional for all types of drivers and governed by the functional specification of the respective driver.

This function is used to retrieve the configuration of the driver. The configuration is returned in the driver control block to which the pointer provided DCBPtr points. For detailed information about the contents of the driver control block, refer to the functional specification of the specific driver.

Some drivers may have a default configuration. If these drivers have not been configured a call of *drv*_GetConfig() returns a pointer to a control block containing this default configuration data and DRV_OK as return value. For drivers that have to be configured, in this case DRV_NOTCONFIGURED is returned.

Call the *drv*_SetConfig() function to configure the driver.

3.5.12 *drv*_Callback – Callback Function of the Driver

Definition:

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

Parameters:

Name	Description
Signal	Pointer to the driver signal (information data)

Return values:

Name	Description
-	-

Description

This function must not be confused with the parameter CallbackFunc passed to *drv*_Init().

This function is only needed for cascaded drivers where the lower layer driver calls the call-back function of the upper layer driver via the frame and thus in general optional. It is the callback entry for the lower layer driver.

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/>