Technical Document - Confidential

GSM PROTOCOL STACK

G23

GSI – GENERAL SERIAL INTERFACE

Document Number:	8434.104.01.002
Version:	0.3
Status:	Draft
Approval Authority:	
Creation Date:	2001-Jun-15
Last changed:	2015-Mar-08 by XINTEGRA
File Name:	Gsi_api.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.

No part of this document may be reproduced or transmitted in any form or by any means, electronically or mechanically, including photocopying and recording, for any purpose without the express written permission of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
2001-Jun-15	STW		0.1		1
2001-Nov-12	STW		0.2		2
2003-May-20	XINTEGRA		0.3	Draft	

Notes:

1. Initial version
2. Better description, correction of type errors

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_8415.0026] 8415.026.99.012; March 19, 1999
Generic Driver Interface – Functional Specification; Condat

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)
- Test and integration support tools.

This document describes the functional interface of the G23 Standard Serial Receiver Transmitter (GSI) driver, API. This driver can be used for common communication purposes and includes UART, USART and IRDA devices. This API of the driver is derived from the generic driver interface specification [C_8415.0026].

NOTE: The driver devices need to be configured after initialization. Only the following functions can be called while the driver is not configured: GSI_Clear(), GSI_SetSignal() and GSI_SetConfig(). All other functions return DRV_NOTCONFIGURED. After the Initialization of the driver the device has to be configured with the GSI_SetConfig() function.

3 Interface description of the GSI driver

3.1 Data types

Name	Description
UBYTE	unsigned 8 bit integer data type
BYTE	signed 8 bit integer data type
USHORT	unsigned 16 bit integer data type
SHORT	signed 16 bit integer data type
ULONG	unsigned 32 bit integer data type
LONG	signed 32 bit integer data type
T_GSI_DCB	Device Control Block

3.1.1 T_GSI_DCB – Device Control Block

Definition:

```
typedef struct T_GSI_DCB
{
  USHORT         Baud
  UBYTE DataBits
  UBYTE StopBits
  UBYTE Parity
  UBYTE RxFlowControl
  UBYTE TxFlowControl
  USHORT         RxBufferSize
  USHORT         TxBufferSize
  USHORT         RxThreshold
  USHORT         TxThreshold
  UBYTE XON
  UBYTE XOFF
  UBYTE EscChar
```

```
    USHORT        GuardPeriod
  } ;
```

Description:

The device control block data type contains all parameters used to configure a serial device. The following table contains a list of the data elements and brief descriptions of them.

Data element	Description
Baud	Transmission rate in bits/sec. See the following corresponding table (Table 1). The GSI_BAUD_AUTO should be used for auto detection of the baud rate and the framing format (if supported by the driver). The expected character for automatic detection is an "A" or "a". After successful detection of the speed and character framing the driver should work with the measured values.
DataBits	Number of bits per character. See the following corresponding table (Table 2).
StopBits	Number of stop bits attached to each character. See the following corresponding table (Table 3).
Parity	Type of Parity checking. See the following corresponding table (Table 4).
RxFlowControl	Type of flow control for receive direction. See the following corresponding table (Table 5).
TxFlowControl	Type of flow control for transmit direction. See the following corresponding table (Table 5).
RxBufferSize	Size of the receiver buffer in bytes.
TxBufferSize	Size of the transmitter buffer in bytes.
RxThreshold	Amount of received bytes that triggers the signal DRV_SIGTYPE_READ.
TxThreshold	Low watermark of the TX buffer that triggers the signal DRV_SIGTYPE_WRITE.
XON	ASCII code of the character which should be detected/send as XON. This parameter should be ignored if non of the flow control modes is set to Software-Flow-Control.
XOFF	ASCII code of the character which should be detected/send as XOFF. This parameter should be ignored if non of the flow control modes is set to Software-Flow-Control.
EscChar	ASCII character which could appear three times as an escape sequence. This parameter should be ignored if escape sequence detection is deactivated.
GuardPeriod	Denotes the minimal duration of the rest before the first and after the last character of the escape sequence, and the maximal receiving duration of the whole escape string. The unit of this parameter should be milliseconds. If this character is set to zero the driver will not detect escape sequences.

3.2 Constants

Name	Description
DRV_OK	Return value indicating the function completed successfully
DRV_INITIALIZED	Device is already initialized
DRV_NOTCONFIGURED	Device is not configured
DRV_BUFFER_FULL	The internal buffer is exhausted
DRV_INPROCESS	The requested function is currently being executed
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid
DRV_SIGFCT_NOTAVAILABLE	The requested event signaling functionality is not available
DRV_INTERNAL_ERROR	Unspecified internal driver error

Possible transmission rate	Description
GSI_BAUD_812500	Transmission rate of 812500 bits/sec.
GSI_BAUD_406250	Transmission rate of 406250 bits/sec.
GSI_BAUD_203125	Transmission rate of 203125 bits/sec.
GSI_BAUD_115200	Transmission rate of 115200 bits/sec.
GSI_BAUD_57600	Transmission rate of 57600 bits/sec.
GSI_BAUD_38400	Transmission rate of 38400 bits/sec.
GSI_BAUD_33900	Transmission rate of 33900 bits/sec.
GSI_BAUD_28800	Transmission rate of 28800 bits/sec.
GSI_BAUD_19200	Transmission rate of 19200 bits/sec.
GSI_BAUD_14400	Transmission rate of 14400 bits/sec.
GSI_BAUD_9600	Transmission rate of 9600 bits/sec.
GSI_BAUD_7200	Transmission rate of 7200 bits/sec.
GSI_BAUD_4800	Transmission rate of 4800 bits/sec.
GSI_BAUD_2400	Transmission rate of 2400 bits/sec.
GSI_BAUD_1200	Transmission rate of 1200 bits/sec.
GSI_BAUD_600	Transmission rate of 600 bits/sec.
GSI_BAUD_300	Transmission rate of 300 bits/sec.
GSI_BAUD_150	Transmission rate of 150 bits/sec.
GSI_BAUD_75	Transmission rate of 75 bits/sec.
GSI_BAUD_AUTO	Automatic detection.

Table 1

Data bits	Description
GSI_CHAR5	Send 5 bits per character.
GSI_CHAR6	Send 6 bits per character.
GSI_CHAR7	Send 7 bits per character.
GSI_CHAR8	Send 8 bits per character.

Table 2

Stop bits	Description
GSI_STOP1	Send 1 stop bit.
GSI_STOP15	Send 1.5 stop bits.
GSI_STOP2	Send 2 stop bits.

Table 3

Parity	Description
GSI_PARITYNO	Don't send a parity bit.
GSI_PARITYODD	Send an odd parity bit.
GSI_PARITYEVEN	Send an even parity bit.
GSI_PARITYSPACE	Send a space for parity bit.

Table 4

Flow Controls	Description
GSI_FLOWNO	No Flow Control.
GSI_FLOWHW	The status lines RTS/CTS (CT105/CT106) are used for Flow Control.
GSI_FLOWSW	The characters XON/XOFF are used for Flow Control.

Table 5

3.3 Functions

Name	Description
GSI_Init	Initialization of the serial communication driver
GSI_Exit	De-initialization of serial communication driver
GSI_Read	Read data received via the serial interface
GSI_Write	Send data via the serial interface
GSI_Look	Read data received via the serial interface, but leave data unchanged
GSI_Clear	Re-initialize all buffers
GSI_Flush	Flush transmit buffer
GSI_SetSignal	Define a signal used to indicated special events
GSI_ResetSignal	Un-define a signal the driver uses to indicate an event
GSI_SetConfig	Set a device configuration
GSI_GetConfig	Retrieve a device configuration
GSI_CallBack	Callback entry for the driver

3.3.1 GSI_Init – Driver Initialization

Definition:

```
USHORT GSI_Init
(
        UBYTE               DeviceNo,
        USHORT              DrvHandle,
        T_DRV_CB_FUNC       CallbackFunc,
        T_DRV_EXPORT**      DrvInfo
);
```

Parameters:

Name	Description
DeviceNo	serial device number
DrvHandle	unique handle for this device
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.
DrvInfo	pointer to the driver parameters (see GDI specification document for a description of T_DRV_EXPORT)

Return values:

Name	Description
DRV_OK	Initialization successful
DRV_INVALID_PARAMS	The specified device does not exist
DRV_INTERNAL_ERROR	Internal driver error
DRV_INITIALIZED	Device already initialized

Description

The function initializes the module and the connected serial device.

The driver stores the **`DrvHandle`** and passes it in the T_DRV_SIGNAL structure of the **`Signal`** parameter to the calling process every time the callback function is called.

The driver exports its properties like its name, the functions to access driver functionality and a bitfield called flags by the parameter **`DrvInfo`**. If the driver does not support this property export it returns NULL in the **`DrvInfo`** parameter.

The function returns DRV_INITIALIZED if the device has already been initialized and is ready to be used or is already in use.

The driver uses the following default configuration until the function GSI_SetConfig() have been called:

- Transmission rate of 19200 bits/sec
- Send 8 bits per character
- Send 1 stop bit
- Don't send a parity bit
- No Flow Control for Rx and Tx direction
- No escape sequence detection

3.3.2 GSI_Exit – Termination of the driver

Definition:

```
void GSI_Exit
(
        UBYTE                   DeviceNo
);
```

Parameters:

Name	Description
DeviceNo	serial device number

Return values:

Name	Description
-	-

Description

The function is called when the device functionality is no longer required. The function “de-allocates” the resources (interrupts, buffers, etc.). The driver terminates the device regardless of any outstanding data to be sent.

3.3.3 GSI_Read - Read data from the driver

Definition:

```
USHORT GSI_Read
(
        UBYTE               DeviceNo,
        void*               Buffer,
        USHORT*             Length,
        ULONG*              State
);
```

Parameters:

Name	Description
DeviceNo	serial device number
Buffer	This parameter points to the buffer wherein the data is to be copied.
Length	On call: Size of Buffer. On return: If the function returns DRV_OK, it contains the number of characters read. Otherwise it contains 0.
State	Line states of the serial connection (see table below)

Return values:

Name	Description
DRV_OK	Function successful.
DRV_INVALID_PARAMS	The specified device does not exist.
DRV_INTERNAL_ERROR	Internal driver error.
DRV_NOTCONFIGURED	The device is not yet configured.

Description

This function reads out data from the driver. It copies received data into a caller provided buffer and it returns the line states. It should always return immediately after copying the data, without waiting for any more data.

The parameter `*Length` contains the size of the buffer in characters. When the function returns the parameter `*Length` contains the number of read characters.

Each copied character is cleared in the buffer of the driver. The driver only keeps the data available when calling the function GSI_Look().

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

The parameter `State` is used in GSI_Read() and in GSI_Write(). The bits of this bit field are:

signal	bitpos	read/write	meaning
SA	31	read/write	state of SA bit (Device ready)
SB	30	read/write	state of SB bit (Data valid)
X	29	read/write	state of X bit (Flow control)
RING	28	write	RING indicator
ESC	27	read	escape sequence detected
DISC	26	read	link disconnected
BRK	25	read/write	break received/to be send
BRKLEN	0-7	read/write	length of the break signal in characters

The ESC bit is set to 1 to indicate a detected escape sequence.

The DISC bit is set to 1 to indicate a disconnected link.

The BRK bit is set to 1 when the driver has received/to send a break signal. The BRKLEN value is only valid if the BRK bit is set to 1.

The bits SA, SB, X and RING are mapped onto the corresponding lines by the driver. They are provided to give the user of the driver a more abstract view of the status lines, particularly of flow control. The mapping depends on the mode of flow control used:

signal		mode of flow control		
		GSI_FLOWNO	GSI_FLOWHW	GSI_FLOWSW
SA	write	CT107 = DSR	CT107 = DSR	CT107 = DSR
	read	CT108 = DTR	CT108 = DTR	CT108 = DTR
SB	write	CT109 = DCD	CT109 = DCD	CT109 = DCD
	read	CT105 = RTS	ON = 0	CT105 = RTS
X	write	ignore	CT106 = CTS	mapped onto XON/XOFF
	read	ON = 0	CT105 = RTS	
RING	write	CT125 = RI	CT125 = RI	CT125 = RI

The SA bit is always mapped onto DSR/DTR. The bit is set to 0 when the device is ready to communicate.

The SB bit is always mapped onto DCD/RTS unless the mode of flow control is GSI_FLOWHW. In this case RTS is used for flow control and the driver returns always 0 (ON) as SB bit. The bit is set to 0 when the data are valid.

The X bit is mapped on CTS/RTS in case of outband flow control or on XON/XOFF in case of inband flow control. When no flow control is used, the driver ignores any setting of the X bit and returns 0 (ON) as X bit. The bit is set to 0 when the device is ready to receive data.

The RING bit is always mapped onto RI. The bit is set to 1 to indicate an incomming call.

Any unused line must be held in ON condition (0) by the driver.

Any unused bits must be set to 0 by the driver.

NOTE: When calling the function with a buffer size of 0, the function stores the number of characters available in the RX buffer of the driver in the parameter **`*Length`**. In this case, **`Buffer`** can be set to NULL and the function will return DRV_OK. The line states will be delivered anyway.

3.3.4 GSI_Write – Transmit data via the serial interface

Definition:

```
USHORT GSI_Write
(
        UBYTE           DeviceNo,
        void*           Buffer,
        USHORT*         Length,
        ULONG           State,
        ULONG           Mask
);
```

Parameters:

Name	**Description**
DeviceNo	serial device number
Buffer	This parameter points to the buffer that is passed to the driver for further processing.
Length	On call: number of characters to write. On return: If the function returns DRV_OK, it contains the number of characters written. Otherwise it contains 0.
State	is a bit field containing the current status bits. Only the lines which are marked with the „set"-access can be used to change the state of this line. The parameter is defined in more detail along with the function GSI_Read().
Mask	is a bit field with the same structure as **State**. Each bit in State corresponds to a bit in **Mask**. Only those status bits are manipulated by the driver, which are marked by a 1 in the **Mask** bit field and which are settable according to the table in the specification of GSI_Read().

Return values:

Name	**Description**
DRV_OK	Function successful.
DRV_INVALID_PARAMS	The specified device does not exist.
DRV_INTERNAL_ERROR	Internal driver error.
DRV_NOTCONFIGURED	The device is not yet configured.

Description

This functions writes data into the driver. It copies the provided data into the buffer of the driver and sets the line state s. This function must return immediately after copying, even if there is not enough space in the driver buffer.

The parameter `*Length` contains the number of characters to write. When the function returns the parameter `*Length` contains the number of written characters.

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

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

NOTE: When calling the function with a buffer size of 0, the function will return the number of characters that can be written into the TX driver buffer in the parameter `*Length`. In this case, `Buffer` can be set to NULL and the function will return DRV_OK. The line states will be set anyway.

3.3.5 GSI_Look – Look at received data of the driver

Definition:

```
USHORT GSI_Look
(
        UBYTE               DeviceNo,
        void*               Buffer,
        USHORT*             Length,
        ULONG*              State
) ;
```

Parameters:

Name	Description
DeviceNo	serial device number
Buffer	This parameter points to the buffer wherein the data is to be copied.
Length	On call: Size of Buffer. On return: If the function returns DRV_OK, it contains the number of characters read. Otherwise it contains 0.
State	Line states of the serial connection (see function GSI_Read())

Return values:

Name	Description
DRV_OK	Function successful.
DRV_INVALID_PARAMS	The specified device does not exist.
DRV_INTERNAL_ERROR	Internal driver error.
DRV_NOTCONFIGURED	The device is not yet configured.

Description

This function reads data from the driver without delete the data in the driver buffer. It copies received data into a buffer, which is provided by the caller, and it returns the line states. It should always return immediately after copying the data, without waiting for any more data.

The parameter **`*Length`** contains the size of the buffer in characters. When the function returns the parameter **`*Length`** contains the number of copied characters.

Each copied character is still held in the buffer of the driver. Consecutive calls to GSI_Look() will lead to identical results, except may be line states. The driver only deletes the data after copy when calling the function GSI_Read().

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

NOTE: When calling the function with a buffer size of 0, the function stores the number of characters available in the RX buffer of the driver in the parameter **`*Length`**. In this case, **`Buffer`** can be set to NULL and the function will return DRV_OK. The line states will be delivered anyway.

3.3.6 GSI_Clear – Clear internal driver buffers

Definition:

```
USHORT GSI_Clear
(
        UBYTE                   DeviceNo,
        USHORT                  BufferType
);
```

Parameters:

Name	Description
DeviceNo	serial device number
BufferType	Bit-mask used to specify which buffer must be cleared

Return values:

Name	Description
DRV_OK	Function successful
DRV_INVALID_PARAMS	The specified device does not exist
DRV_INTERNAL_ERROR	Internal driver error
DRV_INPROCESS	The driver is busy clearing the buffers.

Description

This function is used to clear the device internal buffers. It should always return immediately after changing internal states and resetting internal values, without waiting for any outstanding events.

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 [C_8415.0026]. DRV_BUFTYPE_READ and DRV_BUFTYPE_WRITE can be combined to clear the read and write buffers at once.

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

If the driver could not complete the clearance of the buffers at once the function returns DRV_INPROCESS. The driver will send a signal to the protocol stack when the buffers are cleared completly.

3.3.7 GSI_Flush – Flush internal driver buffer

Definition:

```
USHORT GSI_Flush
(
        UBYTE                   DeviceNo
);
```

Parameters:

Name	Description
DeviceNo	serial device number

Return values:

Name	Description
DRV_OK	Function successful.
DRV_INVALID_PARAMS	The specified device does not exist.
DRV_INTERNAL_ERROR	Internal driver error.
DRV_INPROCESS	The driver is busy flushing the buffer.
DRV_NOTCONFIGURED	The device is not yet configured.

Description

With this function the driver is requested to inform the protocol stack, when all data have been written successfully to the serial device. That means all buffers including buffer in the HW are empty. It considers only TX data. The RX direction has no influence on this function.

This function can be used by the protocol stack to ensure that no more data is to be sent (e.g. before switching between command mode and data mode or before changing the settings of the driver).

In case the TX buffers are already empty or could be written at once, the function returns DRV_OK.

If the driver could not complete flushing the buffers at once the function returns DRV_INPROCESS. The driver will send a signal to the protocol stack when the buffers are flushed completly.

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

3.3.8 GSI_SetSignal – Setup a Signal

Definition:

```
USHORT GSI_SetSignal
(
        UBYTE               DeviceNo,
        USHORT              SignalType
);
```

Parameters:

Name	Description
DeviceNo	serial device number
SignalType	Signal type to be set

Return values:

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

Description

This function is used to define a single or multiple signals that is/are indicated to the process when the event identified in the signal information data type as **`SignalType`** occurs. The driver uses only the standard signals defined in [C_8415.0026].

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

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

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

GSI_Read(): To inform the protocol stack about the availability of data or about a change of the line states, the driver has to call the callback function (provided with the GSI_Init() call) with the **`SignalType`** DRV_SIGTYPE_READ. The values **`UserData`** and **`DataLength`** of the T_DRV_SIGNAL stucture are not used for this signal type. The driver should call this function every time

- when the threshold (**`RxThreshold`** of T_GSI_DCB) is reached or
- when a line status is changed.

If all available data has been read out successfully, the driver returns the number of processed bytes.

```
driver     Callback Function    Protocol Stack
   |             |             |
   |  CallbackFunc     |           |
   |(DRV_SIGTYPE_READ) |           |
   *=====================>     *PSIGNAL      |
   |             |(READY_FOR_READ) |
   |             *=====================>      *
   |             |             |
   |        GSI_Read()         |
   *<=============================================          *
   |       returns No. of Bytes    |
```

If the provided buffer is filled completely then the callback function does not have to be called again with DRV_SIGTYPE_READ even if new data is received by the driver. In this case GSI_Read() must be called again, after space becomes available. A change of the line states however should always be reported with callback.

```
driver     Callback Function    Protocol Stack
   |             |             |
   |  CallbackFunc     |           |
   |(DRV_SIGTYPE_READ) |           |
   *=====================>     *PSIGNAL      |
   |             |(READY_FOR_READ) |
   |             *=====================>      *
   |             |             |
   |        GSI_Read()         |
   *<=============================================          *
   |       returns buffer size     |
   |             |             |
   |             |             |
   |        GSI_Read()         |
   *<=============================================          *
   |       returns No. of Bytes    |
```

GSI_Write(): If all data could be copied successfully, then the return value is equal to the buffer size. In this case the protocol stack may call the function at any time again, when there is new data to be sent.

```
driver     Callback Function   Protocol Stack
   |             |             |
   |        GSI_Write()        |
   *<==============================================         *
   |       returns buffer size     |
   |             |             |
   |       GSI_WriteData()         |
   *<==============================================         *
   |       returns buffer size     |
   |             |             |
```

If the driver can not take over all data then the return value is smaller then the buffer size. In this case the driver must inform the protocol stack, when the amount of buffered data drops below the low watermark (**`TxThreshold`** of T_GSI_DCB). In order to do this the driver has to call the callback function (provided with the GSI_Init() call) with DRV_SIGTYPE_WRITE. Then the function GSI_Write() must be called again by the protocol stack. The values **`UserData`** and **`DataLength`** of the T_DRV_SIGNAL stucture are not used for this signal type.

GSI_Write() may be called by the protocol stack at any time, even if not all data has been copied at the last call and DRV_SIGTYPE_WRITE has not been signalled by the driver yet.

```
driver     Callback Function   Protocol Stack
   |             |             |
   |        GSI_Write()        |
   *<==============================================         *
   |        (No. of Bytes returned           |
   |       < buffer size)          |
   |             |             |
   |  CallbackFunc     |           |
   |(DRV_SIGTYPE_WRITE)|           |
   *====================>      *PSIGNAL      |
   |             |(READY_FOR_WRITE)|
   |             *====================>      *
   |             |             |
   |        GSI_Write()        |
   *<==============================================         *
   |       returns buffer size     |
   |             |             |
```

GSI_Clear(): After all internal buffers are completely cleared the driver calls the callback function (provided with the GSI_Init() call) with DRV_SIGTYPE_CLEAR to inform the protocol stack about the successful completion. The values **`UserData`** and **`DataLength`** of the T_DRV_SIGNAL stucture are not used for this signal type.

```
driver     Callback Function   Protocol Stack
   |             |             |
   |        GSI_Clear()        |
   *<=============================================          *
   |       returns DRV_INPROCESS              |
   |             |             |
   |  CallbackFunc     |           |
   |(DRV_SIGTYPE_CLEAR)|           |
   *=====================>     *PSIGNAL       |
   |             |(BUFFERS_CLEARED)|
   |             *=====================>      *
   |             |             |
```

GSI_Flush(): After all data is completely written to the serial interface the driver calls the callback function (provided with the GSI_Init() call) with DRV_SIGTYPE_FLUSH to inform the protocol stack about the successful completion. The values **`UserData`** and **`DataLength`** of the T_DRV_SIGNAL stucture are not used for this signal type.

The driver must call the callback function only after all data are written and all buffers including in the HW are empty.

```
driver     Callback Function   Protocol Stack
   |             |             |
   |        GSI_Flush()        |
   *<=============================================          *
   |       returns DRV_INPROCESS              |
   |             |             |
   |  CallbackFunc     |           |
   |(DRV_SIGTYPE_FLUSH)|           |
   *=====================>     *PSIGNAL       |
   |             |(DATA_FLUSHED)   |
   |             *=====================>      *
   |             |             |
```

3.3.9 GSI_ResetSignal – Remove a Signal

Definition:

```
USHORT GSI_ResetSignal
(
        UBYTE                   DeviceNo,
        USHORT                  SignalType
);
```

Parameters:

Name	Description
DeviceNo	serial device number
SignalType	Signal type to be reset

Return values:

Name	Description
DRV_OK	Function completed successfully.
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid.
DRV_INTERNAL_ERROR	Internal driver error.
DRV_NOTCONFIGURED	The device is not yet configured.
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available.

Description

This function is used to remove previously set single or multiple signals. The signals that are removed are identified by the SignalType. If the provided SignalType can not be located, the function returns DRV_INVALID_PARAMS.

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

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

3.3.10 GSI_SetConfig – Setup a device configuration

Definition:

```
USHORT GSI_SetConfig
(
        UBYTE                   DeviceNo,
        T_GSI_DCB*              DCBPtr
);
```

Parameters:

Name	Description
DeviceNo	serial device number
DCBPtr	Pointer to the device 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
DRV_INTERNAL_ERROR	Internal driver error

Description

This function is used to configure the serial device (port, transmission rate, flow control, etc). The device can be configured at any time. The parameters that can be configured are included in the device control block T_GSI_DCB. For detailed information about the contents of the device control block, refer to Chapter 3.1.1.

If any value of the configuration is out of range, not supported or invalid in combination with any other value of the configuration, the function returns DRV_INVALID_PARAMS.

Call the GSI_GetConfig() function to retrieve the driver's configuration.

The driver needs to be configured after initialization. Only the following functions can be called while the driver is not configured: GSI_Clear(), GSI_SetSignal() and GSI_SetConfig(). All other functions return DRV_NOTCONFIGURED.

3.3.11 GSI_GetConfig – Retrieve a driver configuration

Definition:

```
USHORT GSI_GetConfig
(
        UBYTE                   DeviceNo,
        T_GSI_DCB*              DCBPtr
);
```

Parameters:

Name	Description
DeviceNo	serial device number
DCBPtr	Pointer to the device control block

Return values:

Name	Description
DRV_OK	Function successfully completed.
DRV_INTERNAL_ERROR	Internal driver error.
DRV_NOTCONFIGURED	The device is not yet configured.

Description

This function is used to retrieve the configuration of the serial device. The configuration is returned in the device control block to which the provided pointer `DCBPtr` points. For detailed information about the contents of the device control block, refer to Chapter 3.1.1.

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

3.3.12 GSI_CallBack – Callback entry for the driver

Definition:

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

Parameters:

Name	Description
Signal	Pointer to the signal information data

Return values:

Name	Description
-	-

Description

This function must not be confused with the parameter CallbackFunc passed to GSI_Init().

This function is only needed for cascaded drivers where the lower layer driver calls the callback function of the upper layer driver via the frame.

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/>